SERIES 60 (LEVEL 68)

MULTICS PROGRAMMER'S MANUAL
COMMANDS AND ACTIVE FUNCTIONS

SUBJECT

Additions and Changes to Standard Multics Commands and Active Functions

SPECIAL INSTRUCTIONS

This is the third revision to AG92, dated January 1979.

This revision replaces Addendum A, B, and C of AG92, Revision 2.

The following commands are obsolete and have been moved to Appendix A in this manual for user convenience.

fs_chname	print_trans_search_rules
mail	set_trans_search_rules

The following commands appear in this manual for the first time:

archive_table	no_save_on_disconnect
convert_characters	save_on_disconnect
hangup	

SOFTWARE SUPPORTED

Multics Software Release 8.0

ORDER NUMBER

AG92-03

December 1979

Honeywell

SERIES 60 (LEVEL 68)

MULTICS PROGRAMMER'S MANUAL COMMANDS AND ACTIVE FUNCTIONS ADDENDUM A

SUBJECT

Additions and Changes to Standard Multics Commands and Active Functions

SPECIAL INSTRUCTIONS

This is the first addendum to AG92, Revision 3, dated December 1979.

Insert the attached pages into the manual according to the collating instructions on the back of this cover.

Note:

Insert this cover after the manual cover to indicate the updating of the document with Addendum A.

SOFTWARE SUPPORTED

Multics Software Release 8.0

ORDER NUMBER

AG92-03A

February 1980

26940
1380
Printed in U.S.A.

Honeywell

COLLATING INSTRUCTIONS

To update the manual, remove old pages and insert new pages as follows:

Remove	Insert
	3-240.1, 3-240.2 3-240.3, 3-240.4 3-240.5, 3-240.6 3-240.7, blank
3-559 through 3-568	3-559 through 3-568
3-619, 3-620	3-619, blank 3-619.1 through 3-619.32 3-619.33, 3-620
4-3, 4-4	4-3, blank 4-3.1, 4-4
	4-18.1, blank
4-19, 4-20	4-19, blank 4-19.1, 4-20

© Honeywell Information Systems Inc., 1980

File No.: 1L13

2/80

AG92-03A

The MPM Commands is organized into four sections. Section 1 contains general information on the use of the manual as well as a description, and definition, of format. Section 2 contains a list of the Multics system commands and active function repertoire, arranged functionally. Section 3 contains descriptions of selected standard Multics system commands and active functions, including the syntax of each, arranged alphabetically. Section 4 describes requests used to gain access to the system.

The MPM Subroutines is organized into three sections. Section 1 contains a list of the subroutine repertoire, arranged functionally. Section 2 contains descriptions of the standard Multics subroutines, including the declare statement, the calling sequence, and usage of each. Section 3 contains the descriptions of the I/O modules.

The MPM Subsystem Writers' Guide is a reference of interest to compiler writers and writers of sophisticated subsystems. It documents user-accessible modules that allow the user to bypass standard Multics facilities. The interfaces thus documented are a level deeper into the system than those required by the majority of users.

Examples of specialized subsystems for which construction would require reference to the MPM Subsystem Writers' Guide are:

- A subsystem that precisely imitates the command environment of some system other than Multics.
- A subsystem intended to enforce restrictions on the services available to a set of users (e.g., an APL-only subsystem for use in an academic class).
- A subsystem that protects some kind of information in a way not easily expressible with ordinary access control lists (e.g., a proprietary linear programming system, or an administrative data base system that permits access only to program-defined, aggregated information such as averages and correlations).

The MPM Peripheral I/O manual contains descriptions of commands and subroutines used to perform peripheral I/O. Included in this manual are commands and subroutines that manipulate tapes and disks as I/O devices.

The MPM Communications I/O manual contains information about the Multics Communication System. Included are sections on the commands, subroutines, and I/O modules used to manipulate communications I/O. Special purpose communications I/O, such as binary synchronous communication, is also included.

Several cross-reference facilities help locate information:

- Each manual has a table of contents that identifies the material (either the name of the section and subsection or an alphabetically ordered list of command and subroutuine names) by page number.
- Each manual contains an index that lists items by name and page number.
- Individual program descriptions reference other programs in the same and other manuals.

CONTENTS

		Page
Section 1	Manual Use and Term Definition.	1-1
	Description of Manual Format	1-1
	General Definition of a Command. . . .	1-3
	General Definition of an Active Function.	1-4
	Examples of Command vs. Active Function Use	1-4
	Errors.	1-4
	General Information.	1-5
Section 2	Reference to Commands and Active Functions.	2-1
	Functional Headings of Commands. . . .	2-1
	Access to the System.	2-1
	Storage System, Creating and Editing Segments	2-1
	Storage System, Segment Manipulation	2-2
	Storage System, Directory Manipulation	2-2
	Storage System, Access Control. . .	2-2
	Storage System, Address Space Control.	2-2
	Formatted Output Facilities	2-3
	Language Translators, Compilers, Assemblers, and Interpreters . . .	2-3
	Object Segment Manipulation	2-3
	Debugging and Performance Monitoring Facilities.	2-3
	Input/Output System Control	2-4
	Command Level Environment	2-4
	Communication Among Users	2-4
	Communication with the System . . .	2-5
	Accounting.	2-5
	Control of Absentee Computations. .	2-5
	Miscellaneous Tools	2-5
	Functional Headings of Active Functions	2-5
	Arithmetic.	2-5
	Character String.	2-6
	Condition Handling.	2-6
	Conversion.	2-6
	Date and Time	2-6

		Page
	Input/Output	2-6
	Logical	2-6
	Miscellaneous	2-7
	Pathname Manipulation	2-7
	Question Asking	2-7
	Storage System Attributes	2-7
	Storage System Names	2-7
	User/Process Information	2-7
Section 3	Commands	3-1
	abbrev (ab)	3-2
	accept_messages (am)	3-7
	add_name (an)	3-10
	add_search_paths (asp)	3-12
	add_search_rules (asr)	3-15
	adjust_bit_count (abc)	3-17
	after (af)	3-19
	and	3-20
	answer	3-21
	apl, v2apl	3-23
	archive (ac)	3-25
	archive_table (act)	3-38
	assign_resource (ar)	3-39
	attach_audit (ata)	3-43
	attach_lv (alv)	3-50
	basic	3-51
	before (be)	3-53
	binary (bin)	3-54
	bind (bd)	3-55
	bool	3-62
	branches, nonlinks	3-64
	nonlinks, see branches	3-64
	calc	3-66
	calendar	3-69
	cancel_abs_request (car)	3-72
	cancel_cobol_program (ccp)	3-75
	cancel_daemon_request (cdr)	3-76
	cancel_resource (cnr)	3-79
	cancel_retrieval_request (crr)	3-80
	canonicalize (canon)	3-82
	ceil	3-84
	change_default_wdir (cdwd)	3-85
	change_error_mode (cem)	3-86
	change_wdir (cwd)	3-87
	check_iacl	3-88
	check_info_segs (cis)	3-89
	close_file (cf)	3-92
	cobol	3-93
	cobol_abs (cba)	3-99
	collate	3-101

Page

collate9. 3-102
compare 3-103
compare_ascii (cpa) 3-105
contents. 3-109
convert_characters (cvc). 3-110
copy (cp) 3-112
copy_acl. 3-115
copy_cards (ccd). 3-116
copy_characters (cpch). 3-118
copy_dir (cpd). 3-119
copy_file (cpf) 3-123
copy_iacl_dir 3-129
copy_iacl_seg 3-130
create (cr) 3-131
create_data_segment (cds) 3-133
create_dir (cd) 3-134
cumulative_page_trace (cpt) 3-137
damaged_sw_off (dsf). 3-140
damaged_sw_on (dsn) 3-141
date. 3-142
date_compiled (dtc) 3-143
date_time 3-145
day 3-146
day_name. 3-147
debug (db). 3-148
decat 3-185
decimal (dec) 3-186
decode. 3-187
default 3-188
default_wdir (dwd). 3-190
defer_messages (dm) 3-191
delete (dl) 3-192
delete_acl (da) 3-194
delete_dir (dd) 3-196
delete_iacl_dir (did) 3-198
delete_iacl_seg (dis) 3-200
delete_message (dlm). 3-202
delete_name (dn). 3-203
delete_search_paths (dsp) 3-205
delete_search_rules (dsr) 3-206
detach_audit (dta). 3-207
detach_lv (dlv) 3-208
directories (dirs). 3-209
directory 3-211
discard_output (dco). 3-212
display_audit_file (daf). 3-213
display_cobol_run_unit (dcr). . . . 3-218
display_pl1io_error (dpe) 3-219
divide. 3-220
do. 3-221

Page

dprint (dp) 3-227
dpunch (dpn). 3-232
dump_segment (ds) 3-236
edm 3-239
encode. 3-241
enter_abs_request (ear) 3-242
enter_retrieval_request (err) . . . 3-248
entries 3-250
entry 3-252
equal 3-253
equal_name (enm). 3-254
exec_com (ec) 3-255
exists. 3-265
expand_cobol_source (ecs) 3-269
fast. 3-272
file_output, fo 3-273
   revert_output, ro. 3-273
   syn_output, so 3-273
   terminal_output, to. 3-273
revert_output, ro SEE file_output. 3-273
syn_output, so SEE file_output . . 3-273
terminal_output, to SEE
 file_output. 3-273
files 3-276
floor 3-278
format_line (fl). 3-279
fortran (ft). 3-281
fortran_abs (fa). 3-281
gcos (gc) 3-283
general_ready (gr). 3-284
get_pathname (gpn). 3-291
get_quota (gq). 3-292
get_system_search_rules (gssr). . . 3-293
greater 3-294
have_mail 3-295
help. 3-296
hexadecimal (hex) 3-316
high. 3-317
high9 3-318
home_dir (hd) 3-319
hour. 3-320
how_many_users (hmu). 3-321
if. 3-323
immediate_messages (im) 3-324
indent (ind). 3-325
index 3-328
index_set 3-329
initiate (in) 3-331
io_call (io). 3-333
last_message (lm) 3-349

Page

last_message_sender (lms) 3-350
last_message_time (lmt) 3-352
length (ln) 3-353
less. 3-354
line_length (ll). 3-355
link (lk) 3-356
links, nonbranches. 3-358
list (ls) 3-360
list_abs_requests (lar) 3-373
list_accessible (lac) 3-378
list_acl (la) 3-380
list_daemon_requests (ldr). 3-382
list_help (lh). 3-386
list_iacl_dir (lid) 3-388
list_iacl_seg (lis) 3-390
list_not_accessible (lnac). 3-392
list_ref_names (lrn). 3-394
list_resource_types (lrt) 3-396
list_resources (lr) 3-397
list_retrieval_requests (lrr) . . . 3-400
logout. 3-404
long_date 3-405
low 3-406
lower_case. 3-407
ltrim 3-408
lv_attached 3-409
manage_volume_pool (mvp). 3-410
master_directories (mdirs). 3-412
max 3-413
memo. 3-414
merge 3-420
merge_ascii (ma). 3-421
min 3-426
minus 3-427
minute. 3-428
mod 3-429
month 3-430
month_name. 3-431
move (mv) 3-432
move_abs_request (mar). 3-435
move_daemon_request (mdr) 3-437
move_dir (mvd). 3-440
move_quota (mq) 3-443
msfs. 3-445
nequal. 3-447
new_fortran 3-448
new_proc. 3-456
ngreater. 3-457
nless 3-458
nonmaster_directories (nmdirs). . . 3-459

	Page
no_save_on_disconnect	3-461
nondirectories (nondirs)	3-462
nonfiles	3-464
nonmsfs	3-466
nonnull_links (nnlinks)	3-468
nonsegments (nonsegs)	3-470
nonzero_files (nzfiles)	3-472
nonzero_msfs (nzmsfs)	3-474
nonzero_segments (nzsegs)	3-476
not	3-478
null_links	3-479
octal (oct)	3-481
on	3-482
or	3-486
overlay (ov)	3-487
page_trace (pgt)	3-488
path	3-490
picture (pic)	3-491
pl1	3-493
pl1_abs (pa)	3-500
plus	3-502
print (pr)	3-503
print_attach_table (pat)	3-505
print_auth_names (pan)	3-507
print_default_wdir (pdwd)	3-508
print_mail (prm)	3-509
print_messages (pm)	3-513
print_motd (pmotd)	3-515
print_proc_auth (ppa)	3-516
print_request_types (prt)	3-517
print_search_paths (psp)	3-518
print_search_rules (psr)	3-519
print_wdir (pwd)	3-520
probe (pb)	3-521
process_dir (pd)	3-547
profile (pf)	3-548
program_interrupt (pi)	3-555
progress (pg)	3-557
qedx	3-560
query	3-568
quotient	3-569
read_mail (rdm)	3-570
ready (rdy)	3-586
ready_off (rdf)	3-587
ready_on (rdn)	3-588
release (rl)	3-589
rename (rn)	3-590
repeat_query (rq)	3-592
reprint_error (re)	3-594
resolve_linkage_error (rle)	3-596

	Page
reserve_resource, rsr	3-597
resource_usage (ru)	3-599
response	3-601
reverse (rv)	3-603
reverse_after (rvaf)	3-604
reverse_before (rvbe)	3-605
reverse_decat (rvdecat)	3-606
reverse_index (rvindex)	3-607
reverse_search (rvsrh)	3-608
reverse_verify (rvverify)	3-609
rtrim	3-610
run	3-611
run_cobol (rc)	3-617
runoff_abs (rfa)	3-620
safety_sw_off (ssf)	3-622
safety_sw_on (ssn)	3-623
save_on_disconnect	3-624
search	3-625
segments (segs)	3-626
send_mail (sdm)	3-628
send_message (sm)	3-648
send_message_acknowledge (sma)	3-650
send_message_express (smx)	3-652
send_message_silent (sms)	3-653
set_acl (sa)	3-654
set_bit_count (sbc)	3-657
set_cc	3-658
set_fortran_common (sfc)	3-660
set_iacl_dir (sid)	3-661
set_iacl_seg (sis)	3-663
set_search_paths (ssp)	3-665
set_search_rules (ssr)	3-666
severity	3-668
slave	4-669
sort	3-670
sort_seg (ss)	3-671
start (sr)	3-675
status (st)	3-676
stop_cobol_run (scr)	3-682
stop_run	3-683
string	3-684
strip	3-685
strip_entry (spe)	3-686
substr	3-687
suffix	3-688
system	3-689
tape_archive (ta)	2-692
terminate (tm)	3-704
time	3-706
times	3-707

	Page
trace	3-708
trace_stack (ts)	3-720
translate	3-725
trunc	3-726
truncate (tc)	3-727
unassign_resource (ur)	3-729
underline	3-730
unique	3-731
unlink (ul)	3-732
upper_case	3-733
user	3-734
verify	3-738
vfile_adjust (vfa)	3-739
vfile_status (vfs)	3-741
volume_dump_switch_off (vdsf)	3-743
volume_dump_switch_on (vdsn)	3-744
walk_subtree (ws)	3-745
where (wh)	3-747
where_search_paths, (wsp)	3-749
who	3-750
working_dir (wd)	3-753
year	3-754
zero_segments (zsegs)	3-755

		Page
Section 4	Access to the System	4-1
	dial (d)	4-2
	enter (e)	4-4
	enterp (ep)	4-4
	hangup	4-7
	hello	4-8
	login (l)	4-9
	logout	4-18
	slave	4-19
	MAP	4-20
	029 and 963	4-21
Index		i-1

PREFACE

Primary reference for user and subsystem programming on the Multics system is contained in six manuals. The manuals are collectively referred to as the Multics Programmers' Manual (MPM). Throughout this manual, references are frequently made to the MPM. For convenience, these references will be as follows:

Document	Referred To In Text As
Reference Guide (Order No. AG91)	MPM Reference Guide
Commands and Active Functions (Order No. AG92)	MPM Commands
Subroutines (Order No. AG93)	MPM Subroutines
Subsystem Writers' Guide (Order No. AK92)	MPM Subsystem Writers' Guide
Peripheral Input/Output (Order No. AX49)	MPM Peripheral I/O
Communications Input/Output (Order No. CC92)	MPM Communications I/O

The MPM Reference Guide contains general information about the Multics command and programming environments. It also defines items used throughout the rest of the MPM and, in addition, describes such subjects as the command language, the storage system, and the input/output system.

(c) Honeywell Information Systems Inc., 1979
File No.: 1L13

SECTION 1

MANUAL USE AND TERM DEFINITION

This section deals with the proper use of this manual, a description of the format used, and a general definition of terms. New users are particularly encouraged to read this section.

DESCRIPTION OF MANUAL FORMAT

Section 2 contains a breakdown by function of the programs described in this manual. Section 3 contains an alphabetized listing of selected standard Multics system commands and active functions. Section 4 contains descriptions of those commands used to gain access to the Multics system.

Each description in this manual closely parallels the info segment format available online by invoking the help command. Each description provides, minimally, the long (and short) name, syntax line, and function of the program. Standard headings, in the order in which they appear when present, are as follows:

SYNTAX AS A COMMAND:
SYNTAX AS AN ACTIVE FUNCTION:
FUNCTION:
ARGUMENTS:
CONTROL ARGUMENTS:
ACCESS REQUIRED:
NOTES:
EXAMPLES:

All headings appear in capital letters followed by a colon (:).

Some descriptions contain unique headings such as "LIST OF KEYS", or "LIST OF REQUESTS".

Syntax lines give the order of required and optional arguments accepted by a command or active function. Optional portions of syntax are enclosed in braces ({}). The syntax for

active functions is always shown enclosed in brackets ([ ]), which are required for active function use. To indicate that a command accepts more than one of a specific argument, the argument name is pluralized (e.g., paths, {paths}, {-control_args}).

NOTE: Keep in mind the difference between a plural argument name that is enclosed in braces (i.e., optional) and one that is not (i.e., required). If the plural argument is enclosed in braces, clearly no argument of that type need be specified. However, if there are no braces, at least one argument of that type must be specified. Thus "paths" in a syntax line could also be written as:
path1 {path2 ... pathn}
The convention of using "paths" rather than the above is merely a method of saving space.

Different arguments that must be specified in pairs are numbered (e.g., xxx1 yyy1 {... xxxn yyyn}). To indicate that the same generic argument must be specified in pairs, the arguments are indicated with letters and numbers (e.g., pathA1 pathB1 {... pathAn pathBn}).

Some of the standard arguments accepted by active functions are:

str — any character string.

num — any character string that represents a number, either decimal or binary. Examples are integers (5, 1024, or 101b), real numbers (1.37 or -10.01b), and floating-point numbers (1.3e+4 or 1010.001e+5b).

dt — any date-time character string in a form acceptable to the convert_date_to_binary_ subroutine described in MPM Subroutines. Examples are "4/25/79 noon est Sun", "November 7", "7:30 pm 10 June 1980", and "midnight".

star_name is any pathname or User_id conforming to the star convention, described under "Star Names" in the MPM Reference Guide.

Arguments, when present, are listed with a brief description and the default value if any. To indicate one of a group of the same arguments, an "i" is added to the argument name (e.g., pathi, User_idi).

The list of control arguments gives the possible values for -control_args in the syntax line. Both the long and short names

are given when both exist. Those that take values (the following argument, as in "-path >udd>m>Foo") generally indicate these values as follows:

STR	any character string; individual command descriptions indicate any restrictions (e.g., must be chosen from specified list; must not exceed 136 characters).
N	number; individual command descriptions indicate whether it is octal or decimal and any other restrictions (e.g., cannot be greater than 4).
DT	date-time character string in a form acceptable to the convert_date_to_binary_ subroutine described in the MPM Subroutines.
ID	numerical request identifier as described in the MPM Reference Guide.
path	pathname of a storage system entry; unless otherwise indicated, it can be either a relative or an absolute pathname.

The lines below are samples of control arguments that take values:

```
-access_name STR, -an STR
-ring N, -rg N
-date DT, -dt DT
-home_dir path, -hd path
```

The "NOTES" section is used to provide additional information and cross-reference with other manuals.

Examples, while not extensive, attempt to provide additional help and insight on the proper use and formatting of commands and active functions. Lines input by the user are preceded by an exclamation mark (!). Examples of command use show the response a user can expect to see on the terminal. Examples of active function use show the return value substituted by the command processor for the active string.

GENERAL DEFINITION OF A COMMAND

Commands are invoked at the beginning of command lines typed to the system, and immediately following unquoted, unbracketed semicolon (;). Some of the uses commands have are displaying information on the user's terminal, archiving data, and compiling programs. Each command has a specific purpose. The default action performed by a command is generally the most common use of

the command. Many commands have optional arguments that refine the actions that are performed. When invoked correctly, most commands either print information or modify storage system entries, but not both.

GENERAL DEFINITION OF AN ACTIVE FUNCTION

Active functions are most frequently used to shorten the amount of typing required to invoke commands. An active function is invoked inside an active string, a string surrounded by brackets ([]), which is replaced by a character string return value before the command line containing it is executed. Active functions are often used in conjunction with the exec_com, abbrev, and do commands to implement command-language macros.

When several commands are specified on a line, the first is executed before active functions in the second are expanded, and so on. Therefore, the execution of a command can affect the values of active functions appearing later in the line.

Examples of Command vs. Active Function Use

Many programs can be invoked as either command or active function. The format of the active function return string is often slightly different from the command's printed output. To illustrate this difference, examples using the status command and active function are shown below. In these examples, and all interactive examples throughout this manual, lines typed by the user are preceded with an exclamation mark (!).

```
! status report1 -nm
  names:    report_first_quarter.runoff
            report1.runoff
            report1
```

versus the corresponding status active function and response:

```
! string [status report1 -nm]
  report_first_quarter.runoff report1.runoff report1
```

Errors

Commands report errors by signalling command_error and printing a message. Messages that do not begin with "Warning:" usually terminate execution of the command, though later commands on the same line are subsequently executed.

Active functions report errors by signalling active_function_error. Default action is to print a message and return to command level. The user should respond by typing:

release

to abort the command line, and then issue a corrected line.

The command_error and active_function_error conditions are further described in the MPM Reference Guide.

GENERAL INFORMATION

Users are encouraged to take advantage of the information available in the manual index. The index alphabetically lists programs by name and subject (i.e., segment, date/time, resource limits, etc.). Cross references among program descriptions help to locate programs applicable to a given task.

SECTION 2

REFERENCE TO COMMANDS AND ACTIVE FUNCTIONS

The Multics commands and active functions documented in this manual are grouped below according to function. All commands and active functions are listed under at least one functional heading. Some commands and active functions are listed under more than one heading. Descriptions appear in Section 3 in alphabetical order.

FUNCTIONAL HEADINGS OF COMMANDS

Access to the System
Storage System, Creating and Editing Segments
Storage System, Segment Manipulation
Storage System, Directory Manipulation
Storage System, Access Control
Storage System, Address Space Control
Formatted Output Facilities
Language Translators, Compilers, Assemblers, and Interpreters
Object Segment Manipulation
Debugging and Performance Monitoring Facilities
Input/Output System Control
Command Level Environment
Communication Among Users
Communication with the System
Accounting
Control of Absentee Computations
Miscellaneous Tools

Access to the System

dial
enter
enterp
login
logout

Storage System, Creating and Editing Segments

adjust_bit_count
compare_ascii
qedx
runoff

canonicalize
edm
indent
program_interrupt
runoff_abs
set_bit_count
sort_seg

Storage System, Segment Manipulation

adjust_bit_count
archive
compare
compare_ascii
copy
copy_file
create
damaged_sw_off
damaged_sw_on
delete
link
merge_ascii
move
set_bit_count
sort_seg
tape_archive
truncate
unlink
vfile_adjust
volume_dump_switch_off
volume_dump_switch_on

Storage System, Directory Manipulation

add_name
cancel_retrieval_request
copy_dir
create_dir
delete_dir
delete_name
enter_retrieval_request
link
list_retrieval_requests
list
move_dir
rename
safety_sw_off
safety_sw_on
status
tape_archive
unlink
vfile_status
volume_dump_switch_off
volume_dump_switch_on

Storage System, Access Control

check_iacl
copy_acl
copy_iacl_dir
copy_iacl_seg
delete_acl
delete_iacl_dir
delete_iacl_seg
list_accessible
list_acl
list_not_accessible
list_iacl_dir
list_iacl_seg
set_acl
set_iacl_dir
set_iacl_seg

Storage System, Address Space Control

add_search_paths
add_search_rules
attach_lv
change_default_wdir
change_wdir
delete_search_paths
print_proc_auth
print_search_paths
print_search_rules
print_wdir
set_search_paths
set_search_rules

delete_search_rules
detach_lv
get_system_search_rules
initiate
list_ref_names
new_proc
print_default_wdir
terminate
terminate_refname
terminate_segno
terminate_single_refname
where
where_search_paths

Formatted Output Facilities

cancel_daemon_request
dprint
dpunch
dump_segment
list_daemon_requests
move_daemon_request
overlay
print
runoff
runoff_abs

Language Translators, Compilers, Assemblers, and Interpreters

apl
basic
bind
cancel_cobol_program
cobol
cobol_abs
create_data_segment
display_cobol_run_unit
expand_cobol_source
fast
format_cobol_source
fortran
fortran_abs
indent
new_fortran
old_fortran
pl1
pl1_abs
profile
qedx
run_cobol
runoff
runoff_abs
set_fortran_common
stop_cobol_run

Object Segment Manipulation

archive
bind
date_compiled

Debugging and Performance Monitoring Facilities

attach_audit
change_error_mode
cumulative_page_trace
debug
display_audit_file
display_pl1io_error
dump_segment
general_ready
page_trace
probe
profile
progress
ready
ready_off
ready_on
repeat_query
reprint_error
resolve_linkage_error
trace
trace_stack

Input/Output System Control

assign_resource
cancel_resource
cancel_daemon_request
close_file
console_output
copy_cards
copy_file
discard_output
display_pl1io_error
dprint
dpunch
file_output
io_call
line_length
list_daemon_requests
list_resource_types
list_resources
print
print_attach_table
print_request_types
reserve_resource
tape_archive
unassign_resource
vfile_adjust
vfile_status

Command Level Environment

abbrev
add_search_paths
add_search_rules
answer
attach_audit
change_default_wdir
change_error_mode
change_wdir
console_output
delete_search_paths
delete_search_rules
display_audit_file
do
exec_com
fast
file_output
if
general_ready
get_system_search_rules
line_length
memo
new_proc
on
print_default_wdir
print_search_paths
print_search_rules
print_wdir
program_interrupt
ready
ready_off
ready_on
release
repeat_query
reprint_error
resolve_linkage_error
run
set_search_paths
set_search_rules
start
stop_run
where_search_paths

Communication Among Users

accept_messages
defer_messages
delete_message
immediate_messages
print_auth_names
print_mail
print_messages
read_mail
send_mail
send_message
send_message_acknowledge
send_message_express
send_message_silent
who

Communication with the System

cancel_retrieval_request
check_info_segs
damaged_sw_off
damaged_sw_on
help
how_many_users
enter_retrieval_request
list_help
list_retrieval_requests
move_abs_request
print_motd
volume_dump_switch_off
volume_dump_switch_on
who

Accounting

get_quota
move_quota
resource_usage

Control of Absentee Computations

cancel_abs_request
cobol_abs
enter_abs_request
fortran_abs
how_many_users
list_abs_requests
move_abs_request
pl1_abs
runoff_abs
who

Miscellaneous Tools

calc
calendar
canonicalize
decode
encode
manage_volume_pool
memo
progress
walk_subtree

FUNCTIONAL HEADINGS OF ACTIVE FUNCTIONS

Arithmetic
Character String
Condition Handling
Conversion
Date and Time
Input/Output
Logical
Miscellaneous
Pathname Manipulation
Question Asking
Storage System Attributes
Storage System Names
User/Process Information

Arithmetic

ceil
divide
floor
max
min
minus
mod
plus
quotient
times
trunc

Character String

after
before
bool
collate
collate9
copy_characters
decat
format_line
high
high9
index
index_set
length
low
lower_case
ltrim
picture
reverse
reverse_after
reverse_before
reverse_decat
reverse_index
reverse_search
reverse_verfiy
rtrim
search
string
substr
translate
underline
unique
upper_case
verify

Condition Handling

on

Conversion

binary
decimal
hexadecimal
octal

Date and Time

date
date_time
day
day_name
hour
long_date
minute
month
month_name
time
year

Input/Output

io_call

Logical

and
equal
exists
greater
less
nequal
ngreater
nless
not
or

Miscellaneous

contents
default

Pathname Manipulation

directory
entry
equal_name
path
strip
strip_entry
suffix

Question Asking

query
response

Storage System Attributes

lv_attached
status

Storage System Names

branches
default_wdir
directories
entries
files
get_pathname
home_dir
links
master_directories
msfs
nondirectories
nonfiles
nonmaster_directories
nonmsfs
nonnull_links
nonsegments
nonzero_files
nonzero_msfs
nonzero_segments
null_links
process_dir
segments
working_dir
zero_segments

User/Process Information

have_mail
last_message
last_message_sender
last_message_time
severity
system
user

SECTION 3

COMMANDS

This section contains descriptions of selected Multics commands and active functions, presented in alphabetical order.

abbrev (ab)

SYNTAX AS A COMMAND:

abbrev

FUNCTION: provides the user with a mechanism for abbreviating parts of (or whole) command lines in the normal command environment.

NOTES: The abbrev command sets up a special command processor that is called for each command line input to the system until abbrev processing is explicitly reverted. The abbrev command processor checks each input line to see if it is an abbrev request line (recognized by a period (.) as the first nonblank character of the line) and, if so, acts on that request. (Requests are described below under "List of Control Requests.") If the input line is not an abbrev request line and abbreviations are included in the line, the abbreviations are expanded once and the expanded string is passed on to the normal Multics command processor. The abbrev command processor is, therefore, spliced in between the listener and the normal command processor. Note that abbreviations are expanded only once; i.e., abbreviations cannot be nested.

The abbrev command is driven by a user profile segment that contains the user's abbreviations and other information pertinent to execution on the user's behalf. The profile segment resides (by default) in the user's home directory. If the profile segment is not found, it is created and initialized with the name Person_id.profile where Person_id is the login name of the user. For example, if the user Washington logs in under the States project, the default profile segment is:

>user_dir_dir>States>Washington>Washington.profile

The profile segment being used by abbrev can be changed at any time with the .u control request (see below) to any profile segment in the storage system hierarchy to which the user has appropriate access. The entryname of a profile segment must have the suffix profile. A new profile segment can be created by specifying a nonexistent segment to the .u control request. The segment is then created and initialized as a profile segment, assuming the user has the necessary access. The user must be careful not to delete or terminate the segment that is

currently being used as his profile unless he first quits out of abbrev by issuing the .q control request (see below).

The user can suppress expansion of a particular string in a command line by enclosing it within quotes ("). To suppress expansion of an entire command line, see the .<space> control request.

A user might want to include the invocation of the abbrev command in a start_up.ec segment so that he is automatically able to abbreviate whenever he is logged in. See the MPM Reference Guide for a definition of start_up.ec.

NOTES ON CONTROL REQUESTS: An abbrev request line has a period (.) as the first nonblank character of the line. An abbrev request line, with the exception of the .s and .<space> requests, is neither checked for embedded abbreviations nor (even in part) passed on to the command processor. If the command line is not an abbrev request line, abbrev expands it and passes it on to the current command processor.

LIST OF CONTROL REQUESTS: The character immediately after the period of an abbrev request line is the name of the request. The following requests are recognized:

.a <abbr> <rest of line>
add the abbreviation <abbr> to the current profile segment. It is an abbreviation for <rest of line>. Note that the <rest of line> string can contain any characters. If the abbreviation already exists, the user is asked whether to redefine it. The user must respond with "yes" or "no". The abbreviation must be no longer than eight characters and must not contain break characters.

.ab <abbr> <rest of line>
add an abbreviation that is expanded only if found at the beginning of a line or directly following a semicolon (;) in the expanded line. In other words, this is an abbreviation for a command name.

.af <abbr> <rest of line>
add an abbreviation to the profile segment and force it to overwrite any previous abbreviation with the same name. The user is not asked whether to redefine the abbreviation.

abbrev (ab)

.abf <abbr> <rest of line>
add an abbreviation that is expanded only at the beginning of a line and force it to replace any previous abbreviation with the same name. The user is not asked whether to redefine the abbreviation.

.d <abbr1> ... <abbrn>
delete the specified abbreviations from the current profile segment.

.f
enter a mode (the default mode) that forgets each command line after executing it. See the .r and .s requests.

.l <abbr1> ... <abbrn>
list the specified abbreviations and the strings they stand for. If no abbreviations are specified, all abbreviations in the current profile segment are listed.

.la <letter1> ... <lettern>
list all abbreviations starting with the specified letters. <letteri> is expected to be a single character. If no letters are specified, all abbreviations in the current profile segment are listed.

.q
quit using the abbrev command processor. This request resets the command processor to the one in use before invoking abbrev and, hence, prevents any subsequent action on the part of abbrev until it is explicitly invoked again.

.r
enter a mode that remembers the last line expanded by abbrev. See the .f and .s requests.

.s <rest of line>
show the user how <rest of line> would be expanded but do not execute it. The .s request with no arguments shows the user the last line expanded by abbrev and is valid only if abbrev is remembering lines. See the .f and .r requests.

.u <profile>
specify to abbrev the pathname of a profile segment to use. <profile segment> becomes the current working profile segment. The user needs "r" access to use the profile segment and "w" access to add and delete abbreviations.

.p
print the name of the profile segment being used.

abbrev (ab)

.<space> <rest of line>
pass <rest of line> on to the current command processor without expanding it. Using this request, the user can issue a command line that contains abbreviations that are not to be expanded.

NOTES ON BREAK CHARACTERS: When abbrev expands a command line, it treats certain characters as special or break characters. An abbreviation cannot contain break characters. Any character string that is less than or equal to eight characters long and is bounded by break characters is a candidate for expansion. The string is looked up in the current profile segment and, if it is found, the expanded form is placed in (a copy of) the command line to be passed on to the normal command processor.

The characters that abbrev treats as break characters are:

newline	
formfeed	
vertical tab	
horizontal tab	
space	
quote	"
dollar sign	$
apostrophe	'
grave accent	`
period	.
semicolon	;
vertical bar	|
parentheses	( )
less than	<
greater than	>
brackets	[ ]
braces	{ }

EXAMPLES: Suppose that a user is typing the segment name suffix fortran repeatedly while editing FORTRAN source segments. The user might wish to abbreviate the suffix to "ft" as follows:

Invoke the abbrev command:

```
! abbrev
```

Define the abbreviation:

```
! .a ft fortran
```

abbrev (ab)

Now that "ft" is defined invoke a text editor to create or edit the source segment:

```
! qedx
! r sample.ft
```

In order to write out one of the segments from qedx by a different name, the user must type the expanded name since the qedx command (and not the abbrev command processor) is intercepting all terminal input. For example, after editing sample.fortran the user might want to write out the changed version as example.fortran. This can be done by typing to qedx:

```
! w example.fortran
```

If instead the user types:

```
! w example.ft
```

a segment is created by exactly that name (example.ft). In this case, if the user tries to print the segment while at command level (by typing "print example.ft"), the abbrev processor expands the command line and the print command looks for a segment named example.fortran; since no such segment exists, the print command responds with an error message.

accept_messages (am)

SYNTAX AS A COMMAND:

am {destination} {-control_args}

FUNCTION: initializes or reinitializes the user's process for accepting messages sent by the send_message command and notifications of the form "You have mail." sent by the send_mail command.

ARGUMENTS:

destination
is of the form Person_id.Project_id to specify a mailbox. The default is the user's default mailbox. If destination contains either < or >, it is assumed to be the pathname of a mailbox.

CONTROL ARGUMENTS:

-brief, -bf
prevents accept_messages from informing the user that it is creating a mailbox and prints messages in short format (see the -short control argument below).

-pathname path, -pn path
specifies a mailbox by pathname. The mbx suffix is assumed. This control argument and the destination argument are mutually exclusive.

-call {cmdline}
when the message is received, instead of printing it in the default format, accept_messages calls the command processor with a string of the form:

cmdline number sender time message {path}

where:

cmdline — is any Multics command line; cmdline must be enclosed in quotes if it contains blanks or other command language characters.

number — is the sequence number of the message, assigned when the -hold control argument is used; otherwise, number is 0.

sender — is the User_id of the person who sent the message.

accept_messages (am)

time is the date-time the message was sent.

message is the actual message sent.

path is the pathname of the mailbox to which the message was sent. If the message was sent to the default mailbox, path is omitted.

To reverse the effect of a previously specified -call control argument, the user can specify the -call control argument with no cmdline argument.

-flush DT
discards messages sent before the specified date-time, where DT is a string acceptable to the convert_date_to_binary_ subroutine (described in the MPM Subroutines). This control argument is intended to be used by operators and consultants.

-hold, -hd
holds messages until explicitly deleted by the delete_message command. Messages printed when the -hold is in effect are preceded by an identifying number.

-long, -lg
precedes every message printed by the sender's Person_id and Project_id. This is the default.

-nohold
reverts -hold

-prefix STR
places the string STR in front of all messages printed as they are received. The string can be up to 12 characters long and can contain the ioa_ control strings ^/ ^| and ^- if desired.

-print, -pr
prints all messages that were received since the last time the user was accepting messages.

-short, -sh
precedes consecutive messages from the same sender by "=:" instead of the Person_id and Project_id.

-time N, -tm N
prints undeleted messages every N minutes, preceded by a message of the form:

You have X messages

where X is the number of undeleted messages. If N equals 0, time mode is reset.

NOTES: The user should not give conflicting control arguments in the same invocation of the command (i.e., -long and -short or -long and -brief).

If the mailbox:

>udd>Project_id>Person_id>Person_id.mbx

does not exist, the accept_messages command creates it. An event channel is created to receive wakeups from send_message so that when a message is received, it is printed on the user's terminal. Messages sent when the user is not logged in or when the user is deferring messages (see the defer_messages command) are saved in the mailbox and can be read later by invoking the print_messages command. The send_mail command stores mail in the same mailbox. See "Extended Access" in the print_mail command description for an explanation of mailbox access.

Channel and process identifiers are stored in the user's mailbox. Since only one process can receive a wakeup when a message is placed in the mailbox, it is not advisable for several users to share the same mailbox.

add_name (an)

SYNTAX AS A COMMAND:

an path names

FUNCTION: adds alternate name(s) to the existing name(s) of a segment, multisegment file, directory, or link.

ARGUMENTS:

path
is the pathname of a segment, multisegment file, directory, or link.

names
are additional names to be added.

ACCESS REQUIRED: modify on the parent directory.

NOTES: The equal and star conventions can be used. See "Constructing and Interpreting Names" in the MPM Reference Guide.

Two entries in a directory cannot have the same entryname; therefore, special action is taken by this command if the added name already exists in the directory that contains the path argument. If the added name is an alternate name of another entry, the name is removed from this entry, added to the entry specified by path, and the user is informed of this action. If the added name is the only name of another entry, the user is asked whether to delete this entry. If the answer is "yes", the entry is deleted and the name is added to the entry specified by path; if the answer is "no", no action is taken.

See also the descriptions of the delete_name and rename commands.

EXAMPLES:

The command line:

```
! an >my_dir>example.pl1 sample.pl1
```

adds the name sample.pl1 to the segment example.pl1 in the directory >my_dir.

The command line:

```
! an >udd>**.private ==.public
```

adds to every entry having a name with private as the last component a similar name with public, rather than private, as the last component.

add_search_paths (asp)

SYNTAX AS A COMMAND:

```
asp search_list search_path1 {-control_args} ...
     search_pathN {-control_args}
```

FUNCTION: adds one or more search paths to the specified search list.

ARGUMENTS:

search_list
is the name of the search list to which the new search paths are added. Synonyms of search_list are described in the individual command descriptions.

search_pathi
specifies a new search path, where search_pathi is a relative or absolute pathname or a keyword. (For a list of acceptable keywords see "List of Keywords" below.) Each search_pathi can be followed by either the -after, -before, -first, or -last control argument to specify its position within the search list. If no search path position control argument is specified, -last is assumed.

CONTROL ARGUMENTS:
are used only with the search_path arguments and can be chosen from the following:

-after STR, -af STR
specifies that the new search path is positioned after the STR search path. The current search path is an absolute or relative pathname or a keyword. In representing STR it is necessary to use the same name that appears when the print_search_paths command is invoked. This control argument is incompatible with -before and -first.

-before STR, -be STR
specifies that the new search path is positioned before the STR search path. This control argument is incompatible with -after and -first.

-first, -ft
specifies that the new search path is positioned as the first search path in the search list. This control argument is incompatible with -after and -before.

add_search_paths (asp)

-last, -lt
specifies that the new search path is positioned as the last search path in the search list.

LIST OF KEYWORDS:

Listed below are the keywords accepted as search paths in place of absolute or relative pathnames. There is no restriction as to the position of any of these keywords within the search list.

-home_dir, -hd
-process_dir, -pd
-referencing_dir, -rd
-working_dir, -wd

NOTES: In addition, a pathname can be specified with the Multics active function [user name] or [user project]. A search path enclosed in quotes is not expanded when placed in the search list. It is expanded when referenced in a user's process. This feature allows search paths to be defined that identify the process directory or home directory of any user.

If a link target does not exist, the search facility continues to search for a matching entryname.

The search facility is composed of the following commands:

add_search_paths, asp
delete_search_paths, dsp
print_search_path, psp
set_search_paths, ssp
where_search_paths, wsp

EXAMPLES:

The command line:

```
! asp translator >udd>Project_id>Person_id>include
```

adds the absolute pathname >udd>Project_id>Person_id>include as a search path. This new search path is positioned as the last search path in the translator search list.

add_search_paths (asp)

The command line:

```
! asp trans <include_files -first
```

adds the absolute pathname represented by the relative pathname <include_files as a search path to the trans search list where trans is a synonym for translator. This new search path is positioned as the first search path in the search list.

The command line:

```
! asp info info_files -after >doc>info
```

adds the absolute pathname represented by the relative pathname info_files as a search path to the info search list. This new search path is positioned in the info search list after the >doc>info search path.

The command line:

```
asp translator >udd>[user project]>incl -be >ldd>include
```

adds the unexpanded pathname >udd>[user project]>incl to the translator search list. This new search path is positioned before the >ldd>include search path.

add_search_rules (asr)

SYNTAX AS A COMMAND:

```
asr path11 {-control_arg path21} ...
           path1n {-control_arg path2n}
```

FUNCTION: allows the user to change object segment search rules dynamically. The search rules to be added can be inserted at any point in the current search rules.

ARGUMENTS:

path1i
is usually a pathname (relative or absolute) representing a directory to be added to the current search rules. It can also be a keyword (see "List of Keywords" below).

path2i
is usually a pathname (relative or absolute) representing a current search rule. It can also be a keyword (see "List of Keywords" below).

CONTROL ARGUMENTS:
must precede the path2i argument.

-before, -be
place path1i before the current search rule identified by path2i.

-after, -af
place path1i after the current search rule identified by path2i.

LIST OF KEYWORDS: In addition to pathnames, both the path1 and path2 arguments accept the keywords:

initiated_segments
referencing_dir
working_dir

LIST OF PATH1 KEYWORDS: The path1 argument also accepts the keywords:

add_search_rules (asr)

home_dir
process_dir
site-defined keywords.

(See the description of the set_search_rules command for an explanation of the site-defined keywords.)

NOTES: If the add_search_rules command is invoked without the control_arg and path2i arguments, the pathname or keyword specified by path1i is appended to the end of the user's current search rules.

Any representation of a current search rule is acceptable for the path2i argument. It is not necessary to use the same name that appears when the print_search_rules command is invoked.

adjust_bit_count (abc)

SYNTAX AS A COMMAND:

abc paths {-control_args}

FUNCTION: sets the bit count of a segment that for some reason does not have its bit count set properly (e.g., the program that was writing the segment got a fault before the bit count was set, or the process terminated without the bit count being set).

ARGUMENT:

paths
are the pathnames of segments and multisegment files. The star convention is allowed.

CONTROL ARGUMENTS:

-character, -ch
set the bit count to the last nonzero character. The default is the last nonzero word.

-chase
chases links when using the star convention. The default is to not chase links when using the star convention.

-long, -lg
print a message when the bit count of a segment is changed, giving the old and new values.

-no_chase
does not chase links when using the star convention. This is the default.

ACCESS REQUIRED: The user must have write access on the segment or multisegment file. Modify on the parent directory is not required.

NOTES: The adjust_bit_count command looks for the last nonzero 36-bit word or (if specified) the last nonzero character in the segment and sets the bit count to indicate that the word or character is the last meaningful data in the segment.

If the bit count of a segment can be computed but cannot be set (e.g., the user has improper access to the segment), the

adjust_bit_count (abc)

computed value is printed so that the user can use the set_bit_count command after resetting access or performing other necessary corrective measures. See the description of the set_bit_count command.

The adjust_bit_count command should not be used on segments in structured files. The vfile_adjust command should be used to adjust inconsistencies in structured files.

SYNTAX AS A COMMAND:

```
af strA strB
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[af strA strB]
```

FUNCTION: returns the string following the first occurrence of strB in strA. If strB does not occur in strA, the null string is returned.

EXAMPLES:

```
! string [after abcdef123def456 def]
  123def456
! string [after abcdef gh]

! string [format_line XY^aZZ [after 1.4596e+17 7]]
  XYZZ
```

and

SYNTAX AS A COMMAND:

```
and tf_args
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[and tf_args]
```

FUNCTION: returns true if all the tf_args are equal to true, otherwise it returns false. If any one of the tf_args does not have the value true or false, an error message is printed.

SYNTAX AS A COMMAND:

answer STR {-control_args} command_line

FUNCTION: provides preset answers to questions asked by another command.

ARGUMENTS:

STR
is the desired answer to any question. If the answer is more than one word, it must be enclosed in quotes. If STR is -query, the question is passed on to the user. The -query control argument is the only one that can be used in place of STR.

command_line
is any Multics command line. It can contain any number of separate arguments (i.e., have spaces within it) and need not be enclosed in quotes.

CONTROL ARGUMENTS:

-brief, -bf
suppresses printing (on the user's terminal) of both the question and the answer.

-query
skips the next answer in a sequence, passing on the question to the user. The answer is read read from the user_io I/O switch.

-then STR
supplies the next answer in a sequence.

-times N
gives the previous answer (STR, -then STR, or -query) N times only (where N is an integer).

NOTES: Answer provides preset responses to questions by establishing an on unit for the condition command_question, and then executing the designated command. If the designated command calls the command_query_ subroutine (described in the MPM Subroutines) to ask a question, the on unit is invoked to supply the answer. The on unit is reverted when the answer command returns to command level. See "List of System

Conditions and Default Handlers" in the MPM Reference Guide for a discussion of the command_question condition.

If a question is asked that requires a yes or no answer, and the preset answer is neither "yes" nor "no", the on unit is not invoked.

The last answer specified is issued as many times as necessary, unless followed by the -times N control argument.

EXAMPLES: To delete the test_dir directory without being interrogated by the delete_dir command, type:

```
! answer yes -bf delete_dir test_dir
```

To automatically see the first three blocks of an info segment named fred.info and then be interrogated about seeing any more blocks, type:

```
! answer yes -times 2 help fred
```

The help command prints the first block, then prints another block every time the user answers yes. In this example, the first three blocks are printed before the user is interrogated.

Sequences of answers are especially useful in exec_coms and absentee jobs. To supply the sequence of answers "yes, no, no, yes", type:

```
! answer yes -then no -times 2 -then yes command_line
```

To supply the sequence of answers "no, ask the user twice, yes, no", type:

```
! answer no -query -times 2 -then yes -then no command_line
```

SYNTAX AS A COMMAND:

apl {workspace_id} {-control_args}

FUNCTION: invokes the APL interpreter, optionally loading a saved workspace.

ARGUMENTS:

workspace_id
is the pathname of a saved workspace to be loaded. The default is to load the user's continue workspace, if any, otherwise to provide a clear workspace.

CONTROL ARGUMENTS:

-terminal_type STR, -ttp STR
specifies the kind of terminal being used. Possible values of STR are:

1050
2741
1030
ARDS
ASCII
BITPAIRED
CORR2741
LA36
TEK4013
TEK4015
TELERAY11
TN300
TYPEPAIRED

This control argument specifies which one of several character translation tables is to be used by APL when reading or writing to the terminal. Since there are several different kinds of APL terminals, each incompatible with the rest, it is important that the correct table be used. Specifying a terminal type to APL changes the terminal type only as long as APL is active. The default depends on the user's existing terminal type (refer to the set_tty command, in MPM Communications I/O, CC92). These terminal types default to the same APL terminal type: 1050, 2741, CORR2741, ARDS, TN300, TEK4013, TEK4015, ASCII, LA36, TELERAY11. All other terminal types default to ASCII. The APL terminal types

BITPAIRED and TYPEPAIRED are generic terminal types that can be used with any APL/ASCII terminal of the appropriate type.

-brief_errors, -bfe
causes APL to print short error messages. This is the default.

-long_errors, -lge
causes APL to print long error messages. The short form of the message is printed, followed by a more detailed explanation of the error.

-user_number N
sets the APL user number (returned by some APL functions) to N. The default is 100.

-check, -ck
causes a compatibility error to occur if a monadic transpose of rank greater than 2, or a residue or encode with a negative left argument is encountered. (The definition of these cases is different in Version 2 APL from Version 1 APL).

-debug, -db
causes APL to call the listener (cu_$cl) upon system errors. This puts the user at a new command level. The default is to remain in APL. This control argument is intended for debugging apl itself.

-no_quit_handler, -nqh
causes APL to ignore the quit condition. The default is to trap all quits within APL.

-temp_dir path, -td path
changes the directory that is used to hold the temporary segments that contain the active workspace to path. The default is to use the process directory.

NOTES: This command invokes the Version 2 APL interpreter, which replaces the obsolete Version 1 APL interpreter.

For a complete description of the APL language, terminal conventions, and directions for converting Version 1 APL workspaces, refer to Multics APL, Order No. AK95.

SYNTAX AS A COMMAND:

ac key archive_path paths

FUNCTION: combines an arbitrary number of separate segments into one single segment. The constituent segments that compose the archive are called components of the archive segment.

ARGUMENTS:

key
is one of the functions listed below under "List of Keywords." The key functions are listed according to their operation.

archive_path
is the pathname of the archive segment to be created or used. The archive suffix is added if the user does not supply it. If the archive segment does not exist for replace and append operations, it is created as described above. The star convention can be used with extraction and table of contents operations.

paths
are the components to be operated on by table of contents and delete operations. For append, replace, update and extract operations, each path specifies the pathname of a segment corresponding to a component whose name is the entryname portion of the pathname. The star and equal conventions cannot be used. (Some operations may not require any path arguments; refer to the specific operation for details.)

LIST OF OPERATIONS: The archive command performs a variety of operations that the Multics user can employ to create new archive segments and to maintain existing ones. The operations are:

Table of contents
print a table of contents of an archive segment.

Append
append components to, or create, an archive segment.

Replace
replace components in, append to, or create an archive segment.

archive (ac)

Update operation
update an archive segment by replacing components with more recently modified ones.

Delete
delete specified components of an archive segment.

Extract
extract components from an archive segment and place them in segments in the storage system.

Each of these general operations can be specialized to perform several functions and, in many cases, can be combined with the copy and deletion features described below. Such combinations give the user extensive control over the maintenance of his archive segments.

LIST OF KEYWORDS:
is one of the key functions listed below according to their operation.

Table of Contents Operation:

t
print the entire table of contents if no components are named by the path arguments; otherwise print information about the named components only. A title and column headings are printed at the top.

tl
print the table of contents in long form; operates like t, printing more information for each component.

tb
print the table of contents, briefly; operates like t, except that the title and column headings are suppressed.

tlb
print the table of contents in long form, briefly; operates like tl, except that the title and column headings are suppressed.

Append Operation:

a
append named components to the archive segment. (The segments

corresponding to the appended components are not affected.) If a named component is already in the archive, a diagnostic is issued and the component is not replaced. At least one component must be named by the path arguments. If the archive segment does not exist, it is created.

ad
append and delete; operates like a and then deletes all segments that have been appended to the archive. If the safety switch is on for any of the corresponding segments, the user is asked whether to delete the segment.

adf
append and force deletion; operates like a and then forces deletion of all segments that have been appended to the archive.

ca
copy and append; operates like a, appending components to a copy of the new archive segment created in the user's working directory.

cad
copy, append, and delete; operates like ad, appending components to a copy of the archive segment and deleting the appended segments.

cadf
copy, append, and force deletion; operates like adf, appending components to a copy of the archive segment and forcibly deleting the segments requested for appending.

Replace Operation:

r
replace components in, or add components to the archive segment. When no components are named in the command line, all components of the archive for which segments by the same name are found in the user's working directory are replaced. When a component is named, it is either replaced or added. If the archive segment does not exist, it is created.

rd
replace and delete; operates like r, replacing or adding components, then deletes all segments that have been replaced or added.

archive (ac)

rdf
replace and force deletion; operates like r and forces deletion of all replaced or added segments.

cr
copy and replace; operates like r, placing an updated copy of the archive segment in the user's working directory instead of changing the original archive segment.

crd
copy, replace and delete; operates like rd, placing an updated copy of the archive segment in the user's working directory.

crdf
copy, replace, and force deletion; operates like rdf, placing an updated copy of the archive segment in the user's working directory.

Update Operation:

u
update; operates like r except that it replaces only those components for which the corresponding segment has a date-time modified later than that associated with the component in the archive. If the component is not found in the archive segment, it is not added.

ud
update and delete; operates like u and deletes all updated segments after the archive has been updated.

udf
update and force deletion; operates like u and forces deletion of all updated segments.

cu
copy and update; operates like u, placing an updated copy of the archive segment in the user's working directory.

cud
copy, update, and delete; operates like ud, placing an updated copy of the archive segment in the user's working directory.

cudf
copy, update, and delete force; operates like udf, placing an updated copy of the archive segment in the user's working directory.

archive (ac)

Delete Operation:

d
delete from the archive those components named by the path arguments.

cd
copy and delete; operates like d, placing an updated copy of the archive segment in the working directory.

Extract Operations:

x
extract from the archive those components named by the path arguments, placing them in segments in the storage system. The directory where a segment is placed is the directory portion of the path argument. The access mode stored with the archive component is placed on the segment for the user performing extraction. If a segment already exists, this operation observes the duplicated name convention in a manner similar to the copy command. If no component names are given, all components are extracted and placed in segments in the working directory. The archive segment is not modified.

xf
extract and delete force; operates like x, forcing deletion of any duplicate names or segments found where the new segment is to be created.

NOTES: The process of placing segments in an archive is particularly useful as a means of eliminating wasted space that occurs when individual segments do not occupy complete pages of storage. Archiving is also convenient as a means of packaging sets of related segments; it is used this way when interfacing with the Multics binder (see the bind command description in this document).

The table of contents operation and the extract operation use the existing contents of an archive segment; the other operations change the contents of an archive segment. A new archive segment can be created with either the append or replace operation. In each of the operations that add to or replace components of the archive, the original segment is copied and the copy is written into the archive, leaving the original segment untouched unless deletion is specified as part of the operation. Use of the various operations is illustrated in the "Examples" at the end of this description.

archive (ac)

The table of contents operation is used to list the contents of an archive segment. It can be made to print information in long or brief form with or without column headings.

The append operation is used to add components to the archive segment and to create new archive segments. When adding to an existing archive, if a component of the same name as the segment requested for appending is already present in the archive segment, a diagnostic message is printed on the user's terminal and the segment is not appended. When several segments are requested for appending, only those segments whose names do not match existing components are added to the archive segment.

The replace operation is similar to the append operation in that it can add components to the archive segment, and therefore, it is also used to create new archive segments. However, unlike the append operation, if a component of the same name as the segment requested for replacing is already present in the archive segment, that component is overwritten with the contents of the segment. When several segments are requested for replacing, those segments whose names do not match existing components are added to the archive segment, as in the append operation.

The update operation replaces existing components only if the date-time modified of a segment requested for updating is later than that of the corresponding component currently in the archive segment. When a segment whose name does not match an existing component of the archive segment is requested for updating, it is not added to the archive segment.

The delete operation is used only to delete components from archive segments. It cannot delete segments from the storage system and is not analogous to the deletion feature described below.

The extract operation is used to create copies of archive components elsewhere in the storage system. The extract operation performs a function opposite to the append operation.

In addition to the operations described above, there are two features, copying and deletion, that can be combined with

certain operations to modify what they do. Since copying and deletion are features and not operations, they cannot stand alone, but must always be combined with those operations that permit their use. The deletion feature is distinct from the delete operation, as noted below.

The copying feature can be combined with the append, replace, update, and delete operations. Since an archive segment can be located anywhere in the storage system, it is occasionally convenient to move the segment during the maintenance process or to modify the original segment while temporarily retaining an unmodified version. When the copying feature is used, the original archive segment is copied from its location in the storage system, updated, and placed in the user's working directory.

The deletion feature can be combined with the append, replace, and update operations to delete segments from the storage system after they have been added to or replaced in an archive segment. The deletion can be forced to bypass the system's safety function, i.e., the user is not asked whether to delete a protected segment before the deletion is performed. (This is analogous to the operation of the delete_force command.) Nothing is deleted until after the archive segment has been successfully updated.

Deletion of segments (deletion feature) is not to be confused with deletion of components from archive segments. The delete operation is a stand-alone function of the archive command that operates only on components of archive segments, deleting them from the archive. The deletion feature, on the other hand, performs deletions only when combined with an operation of the archive command, and then deletes only segments from the storage system after copies of those segments have been added to, or used to update, archive segments.

The archive command can operate in two ways: if no components are named on the command line, the requested operation is performed on all existing components of the archive segment; if components are named on the command line, the operation is performed only on the named components.

The star convention can be used in the archive segment pathname with extract and table of contents operations; it cannot be used with append, replace, update, and delete

archive (ac)

operations. Component names cannot be specified using the star convention. See "Constructing and Interpreting Names" in the MPM Reference Guide for a discussion of the star convention.

No commands other than archive, archive_table, archive_sort, and reorder_archive should be used to manipulate the contents of an archive segment; using a text editor or other command might result in unspecified behavior during subsequent manipulations of that archive segment. See the descriptions of the archive_sort command and the reorder_archive command in the MPM Subsystem Writers' Guide.

Each component of an archive segment retains certain attributes of the segment from which it was copied. These consist of a single name, the effective mode of the user who placed the component in the archive, the date-time the segment was last modified, and the bit count of the segment. In addition, the date-time that the component was placed in the archive segment is maintained. When a component is extracted from an archive segment and placed in the storage system, the new segment is given the name of the component, the bit count of the component, and the mode associated with the component for the user performing the extraction.

The date-time-modified value of a component has a precision of one tenth of a minute. This means that a copy of a component modified less than a tenth of a minute after the archived copy is not updated. Users who update archives in exec_com segments should be aware of this limitation.

The archive command maintains the order of components within an archive segment. When new components are added, they are placed at the end. The archive_sort or reorder_archive commands (described in the MPM Subsystem Writers' Guide) can be used to change the order of components in an archive segment.

The archive command cannot be used recursively. The user is asked a question if the command detects an attempt to use the archive command prior to the completion of its last operation.

Because the replacement and deletion operations are not indivisible, it is possible for them to be stopped before

completion and after the original segment has been truncated. This can happen, for example, if one gets a record quota overflow. When this situation occurs, a message is printed informing the user of what has happened. In this case, the only good copy of the updated archive segment is contained in the process directory.

Archive segments can be placed as components inside other archive segments, preserving their identity as archives, and can later be extracted intact.

When the archive command detects an internal inconsistency, it prints a message and stops the requested operation. For table of contents and extraction operations, it will have already completed requests for those components appearing before the place where the format error is detected.

For segment deletions after replacement requests, if the specified component name is a link to a segment, the segment linked to is deleted. The link is not unlinked.

The archive command observes segment protection by interrogating the user when (unforced) deletion is requested of a segment to which the user does not have write permission. If the user can obtain write permission (i.e., has modify permission on the superior directory) and replies that the segment should be deleted, the segment is deleted.

The archive command refers to the archive segment by full pathname (rather than only the entryname portion) in all printed messages. See "Examples" below.

EXAMPLES: Assume that the user has several short segments and wishes to consolidate them to save space. The working directory, >udd>Project_id>dir_one, might initially look like the following:

```
! list

   Segments = 5, Lengths = 5.

   rw      1  epsilon
   rw      1  delta
   rw      1  gamma
```

archive (ac)

```
rw      1  beta
rw      1  alpha
```

The user creates an archive segment (using the append key) containing four of the five segments.

```
! archive a greek alpha beta gamma delta
  archive: Creating >udd>Project_id>dir_one>greek.archive
```

The working directory then has one more segment (the archive segment), and a table of contents of the new archive segment shows the four components.

```
! list

  Segments = 6, Lengths = 6.

  rw      1  greek.archive
  rw      1  epsilon
  rw      1  delta
  rw      1  gamma
  rw      1  beta
  rw      1  alpha
```

```
! archive tl greek

          >udd>Project_id>dir_one>greek.archive

name                    updated      mode      modified      length

alpha          09/12/74 1435.0 rw  09/12/74 1434.2  441
beta           09/12/74 1435.0 rw  09/12/74 1434.2  257
gamma          09/12/74 1435.0 rw  09/12/74 1434.2  694
delta          09/12/74 1435.0 rw  09/12/74 1434.2  109
```

After changing the segment delta, the user replaces it in the archive segment and appends (using the replace key) the segment epsilon to the archive segment. The user also deletes the component gamma.

```
! archive r greek delta epsilon
  archive: epsilon appended to >udd>Project>dir_one>
           greek.archive

! archive d greek gamma
```

A table of contents now shows a different set of components.

```
! archive t greek

          >udd>Project_id>dir_one>greek.archive

 updated                name

09/12/74 1435.0      alpha
09/12/74 1435.0      beta
09/12/74 1437.5      delta
09/12/74 1437.5      epsilon
```

The user later replaces the component alpha with an updated copy and deletes the storage system segment alpha, causing the updated column of a table of contents to change and a list of the working directory to show one less segment.

```
! archive rd greek alpha

! archive t greek

          >udd>Project_id>dir_one>greek.archive

   updated                          name

   09/12/74 1641.5  alpha
   09/12/74 1435.0  beta
   09/12/74 1437.5  delta
   09/12/74 1437.5  epsilon

! list

   Segments = 5, Lengths = 5.
```

archive (ac)

```
rw      1  greek.archive
rw      1  epsilon
rw      1  delta
rw      1  gamma
rw      1  beta
```

In another directory, >udd>Project>dir_two, which contains a more recent version of the segment alpha, the user copies and updates the archive segment, causing the component alpha to be replaced and the updated archive segment to be placed in the working directory.

```
! archive cu <dir_one>greek
  archive: Copying >udd>Project_id>dir_one>greek.archive
  archive: alpha updated in >udd>Project_id>dir_two>
             greek.archive


! list

  Segments = 2, Lengths = 2.

  rw      1  greek.archive
  rw      1  alpha


! archive t greek

        >udd>Project_id>dir_two>greek.archive

   updated                        name

  09/12/74 1648.3  alpha
  09/12/74 1435.0  beta
  09/12/74 1437.5  delta
  09/12/74 1437.5  epsilon


! ac t <dir_one>greek

        >udd>Project_id>dir_one>greek.archive

   updated             name

  09/12/74 1641.5  alpha
  09/12/74 1435.0  beta
```

archive (ac)

```
09/12/74 1437.5  delta
09/12/74 1437.5  epsilon
```

Notice that the entry in the updated column for the component alpha differs in the two tables of contents. Finally, the user extracts two components into the new working directory, presumably to work on them.

```
! archive x greek beta delta

! list

   Segments = 4, Lengths = 4.

   rw      1  delta
   rw      1  beta
   rw      1  greek.archive
   rw      1  alpha
```

archive_table (act)

SYNTAX AS A COMMAND:

act archive_path {starnames}

SYNTAX AS AN ACTIVE FUNCTION:

[act archive_path {starnames}]

FUNCTION: returns the names of specified archive components in a specified archive segment. Names are returned separated by single spaces.

ARGUMENTS:

archive_path
is the pathname of an archive segment, with or without the archive suffix. The star convention is NOT allowed.

LIST OF OPTIONAL ARGUMENTS:

starnames
are optional component names to be matched against names of archive components. The star convention is allowed.

NOTES: Invoked as a command, archive_table prints the component names, one name per line.

SYNTAX AS A COMMAND:

ar resource_type {-control_args}

FUNCTION: calls the resource control package (RCP) to assign a resource to the user's process.

ARGUMENT:

resource_type
specifies the type of resource to be assigned. Currently, only device types can be specified. The -device control argument is used to name a specific device to assign. Other control arguments are used to specify characteristics of the device to be assigned. The following device type keywords are supported:
tape_drive
disk_drive
console
printer
punch
reader
special

CONTROL ARGUMENTS:

-device STR, -dv STR
specifies the name of the device to be assigned. If this control argument is specified, other control arguments that specify device characteristics are ignored. (See "Examples" below.) If the -long control argument (see below) is used in conjunction with this control argument, a message containing the name of the assigned device is printed on the user's terminal; otherwise, no message is printed.

-model N
specifies the device model number characteristic. Only a device that has this model number is assigned. In order to find the model numbers that are acceptable, use the print_configuration_deck command described in System Tools, Order No. AZ03.

-track N, -tk N
specifies the track characteristic of a tape drive. The value can be either 9 or 7. If this control argument is not specified and if the -volume control argument is not specified, a track value of 9 is used when assigning a tape device.

assign_resource (ar)

-density N, -den N
specifies the density capability characteristic of a tape drive. There can be more than one instance of this argument. A tape drive is assigned that is capable of being set to all of the specified densities. The acceptable values for this argument are:

200
556
800
1600
6250

Note that the values permitted depend on the particular hardware on the system.

-train N, -tn N
specifies the print train characteristic of a printer.

-line_length N, -ll N
specifies the line length of a printer. Its value must be one that is found in the "line length" field of a printer PRPH configuration card. If this field is not specified on a printer PRPH configuration card, this device characteristic is ignored for this printer.

-volume STR, -vol STR
specifies the name of a volume. If possible, the device assigned is one on which this volume has already been placed. If this is not possible (e.g., the volume is on a device assigned to a process) any available, appropriate, and accessible device will be assigned.

-number N, -nb N
specifies the number of resources to assign. All of the resources assigned have the device characteristics specified by any other arguments passed to this command. If this control argument is not specified, one resource is assigned.

-comment STR, -com STR
is a comment string that is displayed to the operator when the resource is assigned. If more than one string is required, the entire string must be in quotes. Only printable ASCII characters are allowed. Any unprintable characters (also tabs or new lines) found in this string are converted to blanks.

-long, -lg
specifies that all of the device characteristics of the assigned device should be printed. If this argument is not supplied, only the name of the assigned device is printed.

assign_resource (ar)

-system, -sys
specifies that the user wants to be treated as a system process during this assignment. If this argument is not specified or if the user does not have the appropriate access, then the RCP assumes that this assignment is for a nonsystem process.

-wait {N}, -wt {N}
specifies that the user wants to wait if the assignment cannot be made at this time because the resources are assigned to some other process. The value N specifies the maximum number of minutes to wait. If N minutes elapse and a resource is not yet assigned, an error message is printed. If N is not specified, it is assumed that the user wants to wait indefinitely.

-speed N
specifies the speed of a tape drive. The acceptable values depend on the particular hardware on the system and can be the following:
75
125
200

NOTES: Currently, only device resources can be assigned. An assigned device still must be attached by a call to some I/O module. If a device is successfully assigned, the name of the device is printed. (If the user requests a specific device that is successfully assigned, the name of the device is not printed unless the user asks for it. See the -device and -long control arguments above.)

EXAMPLES: In the example below, the user issues the assign_resource command with the "tape_drive" keyword and the -model control argument. The system responds with the name of the assigned device.

```
! assign_resource tape_drive -model 500

   Device tape_04 assigned
```

assign_resource (ar)

In the next example, the user issues the assign_resource command with the "tape_drive" keyword and the -device and -long control arguments. The system responds with the name of the assigned device and the model number, track, density and speed characteristics.

```
! assign_resource tape_drive -device tape_05 -long

   Device tape_05 assigned
   Model     =  500
   Tracks    =  9
   Densities =  200 556 800 1600
   Speed     =  125
```

attach_audit (ata)

SYNTAX AS A COMMAND:

ata {old_switch {new_switch}} {-control_args}

FUNCTION: sets up a specified I/O switch, with a stream_input_output opening, to be audited by the audit_ I/O module.

ARGUMENTS:

old_switch
is the name of an I/O switch to be audited. The default is user_i/o. If only one switch is specified, it is the old_switch.

new_switch
is the name of an I/O switch to be used by the audit_ I/O module. If only one switch argument is given, it is the old_switch. The default value for new_switch is audit_i/o.<time>, where <time> has the value MM/DD/YY--hhmm.m.

CONTROL ARGUMENTS:

-truncate, -tc
truncates the audit file if it already exists. If this control argument is not given, the audit file is extended by default.

-pathname path, -pn path
specifies that path is the pathname of the audit file to use. If pathname is not given, the audit file is in the user's home directory and named date.audit.

NOTES: If used with no arguments, attach_audit sets up auditing for the user_i/o I/O switch with input and output audited and editing on.

Auditing of old_switch is done by moving the attachment of old_switch to new_switch and then attaching old_switch to new_switch via audit_. See the MPM Subroutines discussion of the audit_ I/O module and the MPM Commands discussion of detach_audit for more information.

LIST OF AUDITING REQUESTS:
A three-character sequence is used to make an auditing

attach_audit (ata)

request: the audit trigger character ("!" by default), followed by the specific request character, followed by a newline. An auditing request can either be alone on a line or have text preceding it on the same line.

!.
prints the combination of input and/or output being audited.

!?
prints a brief description of available auditing requests.

!e
enters the audit editor. The entry preceding this sequence becomes the current line to be edited.

!E
enters the audit editor, and processes any text preceding the sequence on the same line as editing requests. If no text precedes the sequence, the effect is the same as for !e.

!a
expands abbreviations in the input line. (See the abbrev command in this manual for more information.)

!r
redisplays the input line and strips off the newline. Further input can then be appended to the redisplayed line until another newline is typed, but no further erase or kill processing is performed on the redisplayed portion. The redisplayed line plus the appended input (if any) becomes the input line that is returned to the I/O module being audited.

!t
instructs the audit_ I/O module not to log the input line; this makes the input line transparent.

!d
specifies that the input line to which this is appended is deleted. This is used to kill a line that has been redisplayed with the !r request.

!n
specifies no operation; this is useful when the !n follows another auditing request sequence that the user does not want interpreted.

NOTES ON AUDIT FILE:

The audit file, by default, has the pathname:

>udd>Project_id>Person_id>date.audit

where date is the first eight characters (the date portion) returned by the date_time_ subroutine at the time of attaching, and is of the form "MM/DD/YY". This pathname can also be specified using active functions:

[home_dir]>[date].audit

The default audit file size is unlimited, and the audit file can become a multisegment file.

The audit editor operates on entries, rather than lines, and the entry type identifiers are:

EL edit line

IC input characters

IL input line

OC output characters

TC trace of control operations

TM trace of modes operations

NOTES ON AUDIT EDITOR: The audit editor is invoked by typing the e or E auditing request sequence described above. It edits and executes lines that have been logged by the audit_ I/O module. The syntax of editing requests is similar to that of qedx requests (see the qedx command in this manual). Any number of requests can be on the same line; spaces are ignored.

Addressing is done the same way as in the qedx editor, with two exceptions. The "." is a request for self-identification rather than an indicator for the current entry, and addresses are expressed in terms of entries in the audit file rather than lines in a buffer. The edit buffer contains only one entry at a time. If the default search tag is in use, as is the case unless specifically overridden, the absolute entry number refers to the number of entries, with the default search tag, from the beginning of the file. Similarly, a relative entry address refers to the number of entries, with the default search tag, before or after the current address.

attach_audit (ata)

LIST OF EDITING REQUESTS: The audit editor requests are presented below in two categories: familiar (qedx-like) requests, and special requests.

s/REGEXP/STR/
substitutes the string STR for occurrences of the regular expression REGEXP in the edit buffer.

ADR
locates the entry with address ADR. If ADR is not followed by a request, the audit file entry is printed. An ADR can contain an absolute entry reference at its beginning, relative addresses in any portion, and regular expressions in any portion. If a regular expression in the address is preceded by the less than character (<), a backward search is done to find a match for the regular expression. An absolute address is either a number, or the dollar sign ($) to indicate the last entry in the audit file.

{ADR1,ADR2}p
prints the current entry if no ADR is specified; prints the addressed audit file entry if a single address is specified; prints entries from address 1 through address 2 if two addresses are specified.

..STR
passes the string STR to the command processor and then returns to the audit editor.

q
quits the editor and returns the current line to the I/O module being audited, with the !e or !E sequence included.

.expand
expands abbreviations in the edit buffer (see the abbrev command in the MPM Commands for a discussion of expansion of abbreviations).

.off
disables auditing of input and output in the editor.

.on
enables auditing of input and output in the editor.

.l
addresses the last audit file entry returned by the audit editor.

.r[STR]
quits the editor and returns the string STR to the I/O module

being audited. If STR is not specified, the r request quits the editor and returns the edit buffer.

.n
returns a newline character.

.type
prints the audit file entry type of the current position.

.exec
passes the edit buffer to the command processor and returns to the audit editor.

.d/STR/
sets the default search tag to the string STR. If STR is only one character, only the first character of the tag is used to determine if an entry is seen (in counting entries and doing searches). If STR is two characters, the match is made on both characters of the tag.

.?
prints a brief description of available audit editor requests.

:
overrides the default search tag for those requests following on the same line (i.e., any tag is matched). A newline reestablishes the default search tag.

The REGEXP field of a substitute request is interpreted as a qedx-style regular expression. The STR field of a substitute request is also interpreted as in qedx, and the & convention is supported. If REGEXP is null in a substitute request, the last REGEXP specified in a previous substitute request is used.

No lines in the audit file are changed by the editor; only copies are modified.

If the audit editor is being audited, the audit editor can be invoked from within the editor. For every level of the editor, a distinct last returned line is remembered.

EXAMPLES: In the example given below, there has been such extensive use of the erase character that the user may want to see it displayed. In order to verify the input line given, it

attach_audit (ata)

can be replayed by using the !r request. The ! at the beginning of the line indicates lines typed by the user.

```
!  str#ty =#-print_mod########modes red!r
   stty -pmodes red
```

This line does not end with a newline character, so the next character typed would appear immediately following the "red" and on the same line. In this example, -pmodes was entered instead of -modes. Typing the following on that same line:

```
!                  ##########modes red!r
```

does not correct the error, but returns:

```
   stty -pmodes redmodes red
```

The erase character cannot be used to correct portions of a line that has already been replayed. The current situation can be corrected as follows:

```
!  stty -pmodes redmodes red!e
!  p
   stty -pmodes redmodes red
!  s/redmodes red/red/ s/pmodes/modes/p
   stty -modes red
!  .r
```

The above procedure enters the audit editor with the !e request. The p request prints the contents of the edit buffer. If no argument is given for p, the most recent input line is printed. Corrections are made to the line and the modified line is printed. The request .r exits the audit editor and returns the line to the I/O module being audited.

An alternative procedure is the following:

```
!  stty -pmodes redmodes red!n
!  stty -modes red
```

The request !n suppresses the entire input line and it is then reentered correctly.

In the first example given, there are two ways to set the red shift mode. It can be turned off and then on again, as follows:

```
!  stty -modes ^red
!  .l.r!E
```

The .l.r enters the audit editor. This puts the last entry returned by the audit editor in the edit buffer, then returns the contents of the buffer. To request the "stty -modes ^red" command, type:

```
!  </^stty/p.r!E
   stty -modes ^red
```

This does a backward search in the audit file for an input entry beginning with stty, puts this entry in the edit buffer, prints the contents of the edit buffer, and returns the contents of the edit buffer.

To see the last five input entries in the audit file at this point, type:

```
!  -4,p!E
   s/redmodes red/red/ s/pmodes/modes/p
   .r
   stty -modes ^red
   .l.r!E
   </^stty/p.r!E
```

To see the last five output entries prior to this invocation of the audit editor, type:

```
!  .d/0/
!  -4,p
   stty -pmodes red stty -pmodes redmodes red
   stty -pmodes redmodes red
   stty -modes red
   stty -modes ^red
```

Note that the entries that are the result of a replay (!r) do not end in a newline character, so they run together on the same line when being printed.

attach_lv (alv)

SYNTAX AS A COMMAND:

alv volume_name

FUNCTION: calls the resource control package (RCP) to attach a logical volume.

ARGUMENTS:

volume_name
specifies the name of the volume to be attached.

ACCESS REQUIRED: A user must have rw access to the logical volume to be attached, as defined by the access control segment (ACS) associated with the logical volume.

NOTES: Attaching a logical volume involves informing the storage system that a particular volume is attached for a particular process. A logical volume (unless it is a public logical volume) must be attached for each process that wishes to use it. To be attached, the logical volume must first be physically mounted. This mounting involves mounting all of the physical volumes that compose the logical volume.

If the specified volume is not already mounted, the system operators are requested to mount the volume, if appropriate resources are available. The attach_lv command does not return until the volume is mounted or the operator has denied the request.

The status command issued with the -device control argument prints the name of the logical volume on which a segment resides.

SYNTAX AS A COMMAND:

basic path {-control_arg}

FUNCTION: invokes the BASIC compiler to translate a segment containing BASIC source code. Either the compiled code is executed, or a standard object segment is created to be executed at a later time.

ARGUMENTS:

path
is the pathname of the segment to be translated. The basic suffix need not appear as part of the pathname. It must, however, be the last component of the name of the source segment.

CONTROL ARGUMENTS:

-compile
requests BASIC to compile the program and generate a bindable Multics standard object segment. The resulting object segment is placed in the user's working directory.

-time N
where N is a decimal number that requests a limit of N seconds on the execution of the BASIC program. If the limit is exceeded, the user is asked whether to continue.

NOTES: The -compile and -time control arguments are incompatible.

If the -compile control argument is not specified, the compiled code is then executed and not saved for future execution. If the -compile control argument is specified, a standard object segment is created for subsequent execution.

For a description of the BASIC language on the Multics system, consult the Multics BASIC manual, Order No. AM82.

For information on using the FAST subsystem to compile BASIC source code, refer to the Multics FAST Subsystem Users' Guide, Order No. AU25.

basic

For a description of the features common to all Multics programming languages, see "Programming Languages" in the MPM Reference Guide. A description of object segments can be found in the MPM Reference Guide in "Creating an Object Segment."

before (be)

SYNTAX AS A COMMAND:

```
be strA strB
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[be strA strB]
```

FUNCTION: returns the string preceding the first occurrence of strB in strA. If strB does not occur in strA, the entire string strA is returned.

EXAMPLES:

```
! string [before abcdef123defabc def]
  abc
! string [before abcdef g]
  abcdef
! string [before abcdef123 abc]

! string [format_line XY^aZZ [before 1.4596e+17 7]]
  XY1.4596e+1ZZ
```

binary (bin)

SYNTAX AS A COMMAND:

bin values

SYNTAX AS AN ACTIVE FUNCTION:

[bin values]

FUNCTION: returns one or more values in binary.

ARGUMENTS:

value
is a value to be processed. The last character of value indicates its type. Acceptable types are binary (b), quartenary (q), octal (o), hexadecimal (h), and unspecified (u). Any valid PL/I real value is allowed. The absence of any specifier means decimal. The unspecified value is limited to 8 characters.

EXAMPLES:

```
! string [binary 657.4o]
110100111.1
```

SYNTAX AS A COMMAND:

bind paths {-control_args}

FUNCTION: produces a single bound object segment from one or more unbound object segments, stored in archive segments, which are called the components of the bound segment.

ARGUMENTS:

paths
are the pathnames of archive segments containing one or more component object segments to be bound. The archive suffix is assumed. Up to 16 input archive segments can be specified. They are logically concatenated in a left-to-right order to produce a single sequence of input component object segments.

CONTROL ARGUMENTS:

-update paths, -ud paths
indicates that the following list of archive segments (paths) specifies update rather than input object segments. The archive suffix is assumed in paths. Up to a combined total of 16 input and update segments can be specified. The contained update object segments are matched against the input object segments by object segment name. Matching update object segments replace the corresponding input object segments; unmatched ones are appended to the sequence of input object segments. If several update object segments have the same name, only the last one encountered is bound into the bound segment.

-list, -ls
produces a listing segment whose name is derived from the name of the bound object segment plus a suffix of list. The listing segment is generated for the purpose of dprinting; it contains the bound segment's bind control segment (see "Notes on Bindfile" below), its bind map, and that information from the bound object segment printed by the print_link_info command. This control argument cannot be invoked with -map. In the absence of the -list or -map control arguments, no listing segment is generated.

-map
produces a listing segment (with the suffixes list and map) that contains only the bind map information. This control argument is incompatible with -list. In the absence of the

bind (bd)

-list or -map control arguments, no listing segment is generated.

-brief, -bf
suppresses printing of warning messages.

NOTES: Compilers and the assembler produce unbound object segments. Binding has three benefits: the reduction of storage fragmentation, the prelinking of external references between the components, and the reduction of size of address space necessary to execute the components.

Each of these benefits saves CPU time and storage usage if the set of components bound is used with regularity. This reduction in usage translates directly into lower charges for the users of the bound segment. System efficiency is also increased by binding together common sets of programs. A reference in one component to an entrypoint defined in another component is resolved during the binding. This prelinking avoids the cost of dynamic linking, and it also ensures that the reference is linked to the component regardless of the state of a process at the moment that dynamic linking takes place. References to an entrypoint are prelinked unless the contrary is specified by an instruction in the bindfile. The bindfile is a segment containing instructions that control various aspects of the binding operation (see "Notes on Bindfile" below). (See the description of the print_link_info command in the MPM Subsystem Writers' Guide.)

NOTES ON OUTPUT: The binder produces as its output two segments: an executable bound object segment and an optional, printable ASCII listing segment. The name of the bound segment is, by default, derived from the entryname of the first input archive segment encountered by stripping the archive suffix from it. The name of the listing segment is derived from the name of the bound segment by adding the list suffix to it. Use of the Objectname master statement in the bindfile (see "List of Master Keywords" below) allows the name of the bound segment to be stated explicitly. In addition, use of the Addname master statement in the binding instructions causes additional segment names to be added to the bound segment. The primary name of the bound segment must not be the same as the name of any component.

NOTES ON BINDFILE: The bindfile is a segment containing symbolic instructions that control the operation of the binder. Its

entryname must contain the bind suffix and it must be archived into any one of the input archive segments (at any location within that archive segment) where it is automatically located and recognized by the binder.

In case two bindfiles are specified, one in an input archive segment and the other in an update archive segment, the latter takes precedence and a warning message is printed to that effect.

The syntax of the bindfile statements consist of a keyword followed by zero or more parameters and then delimited by a statement delimiter. Master statements pertain to the entire bound object segment; normal statements pertain to a single component object within the bound segment. Master statements are identified by master keywords that begin with a capital letter; normal keywords begin with a lowercase letter. A keyword designates a certain action to be undertaken by the binder pertaining to parameters following the keyword.

LIST OF BINDFILE DELIMITERS:

: keyword delimiter
It is used to identify a keyword followed by one or more parameters. A keyword that is followed by no parameters is delimited by a statement delimiter.

; statement delimiter.

, parameter delimiter
(Note, the last parameter is delimited by a statement delimiter).

/* begin comment

*/ end comment

LIST OF MASTER KEYWORDS:

Objectname
the parameter is the segment name of the new bound object.

Order
the parameters are a list of objectnames in the desired binding order. In the absence of an order statement, binding is done in the order of the input sequence. The order

bind (bd)

statement requires that there be a one-to-one correspondence between its list of parameters and the components of the input sequence.

Force_Order
same as Order, except that the list of parameters can be a subset of the input sequence, allowing the archive segments to contain additional segments that are not to be bound (e.g., source programs).

Global
the parameters can be either retain, delete, or no_link. The parameter selected pertains to all component object segments within the bound segment. A global or explicit statement concerning a single component object or a single external symbol of a component object overrides the Global statement for that component object or symbol.

Addname
the parameters are the symbolic names to be added to the bound segment. If Addname has no parameters, it causes the segment names and synonyms of those component objects for which at least a single entrypoint was retained to be added to the bound segment.

No_Table
does not require parameters. It causes the symbol tables from all the component symbol sections containing symbol tables to be omitted from the bound segment. If this keyword is not given, all symbol tables are kept.

Perprocess_Static
does not require parameters. It causes the perprocess_static flag of the bound segment to be turned on, which prevents the internal static storage from being reset during a run unit.

If no bindfile is specified, the binder assumes default parameters corresponding to the following:

Objectname: segment name of the first input archive file.

Global: retain; /*regenerate all definitions*/

LIST OF NORMAL KEYWORDS:

objectname
the single parameter is the name of a component object as it appears in the archive segment. The objectname statement

bind (bd)

indicates that all following normal statements (up to but not including the next objectname statement) pertain to the component object whose name is the parameter of the objectname statement.

synonym
the parameters are symbolic segment names declared to be synonymous to the component object's objectname. When b is declared to be a synonym for a, references (in the bound components) of the form b or b$x (any x) are resolved during binding by searching for a definition of b or x in component a. A synonym instruction must be given if such references are to be prelinked. The synonym instruction also affects dynamic linking so that if b is a reference name for the bound segment, then references of the form b or b$x are resolved by searching component a. In this case, the synonym instruction may reduce the cost of dynamic linking, and it avoids possible ambiguities when two components contain definitions for the symbol b. Users should take care to state explicitly in a synonym statement all the normally used segment names of a component object. For example, the create and create_dir commands are implemented in one procedure, and both have abbreviations; thus a bindfile for the bound segment in which this procedure resides contains:

```
objectname:    create;

synonym:       create, cr, create_dir, cd;
```

Failure to state segment names results in inefficient linker performance.

retain
the parameters are the names of entrypoints defined within the component object segment that the user wishes to retain as entrypoints of the bound object segment.

delete
the parameters are the names of entrypoints defined within the component object segment that the user does not wish to be retained as entrypoints of the new bound segment.

The retain and delete statements are considered exclusive. An error message is displayed if the binder recognizes that two or more such statements were made regarding any single entrypoint.

no_link
the parameters are the names of entrypoints that are <u>not</u> to be

bind (bd)

prelinked during binding. The no_link statement implies a retain statement for the specified names.

global
the parameter can be either retain, delete, or no_link. The parameter selected becomes effective for all entrypoints of the component object. An explicit retain, delete, or no_link statement concerning a given entrypoint of the component object overrides the global statement for that specific entrypoint. A global no_link causes all external references to the component object to be regenerated as links to entrypoints; this allows execution-time substitution of such a component by a free standing version of it, for example for debugging purposes.

table
does not require parameters. It causes the symbol table for the component to be retained and is needed to override the No_Table master keyword, described above.

NOTES ON ERROR MESSAGES: The binder produces three types of error messages. Messages beginning with the word "Warning" do not necessarily represent errors, but warn the user of possible inconsistencies in the input components or bindfile. Messages beginning with the word "binder_" normally represent errors in the input components. Errors detected during the parsing of the bindfile have the format:

Bindfile Error Line #N ....

where N is the line number of the erroneous statement. If an error is detected during parsing, the binder aborts because it cannot bind according to the user's specifications.

The message:

"binder_: Fatal error has occurred; binding unsuccessful."

indicates that it was impossible for the binder to produce an executable object segment because of errors detected during binding. The bound object segment is left in an unpredictable state.

EXAMPLES: The bindfile for the debug command, which is named bound_debug.bind, is as follows:

bind (bd)

```
Objectname:  bound_debug;
Global:      delete;     /*delete all old definitions*/

Addname;                 /*add names debug, db, list_arg_
                           and gr_print to bound segment
                           bound_debug_*/

objectname:  debug;
synonym:     db;         /*indicate db is synonymous to debug*/
retain:      debug,
             db;         /*retain entrynames debug$debug and
                           debug$db*/

objectname:  list_arg_;
retain:      list_arg_; /*retain entryname list_arg_$list_arg_*/
objectname:  gr_print;
retain:      gr_print;  /*retain entryname gr_print$gr_print*/
```

The following illustrates other uses of the bindfile:

```
Objectname:  bound_test;
Global:      delete;     /*delete all old definitions*/
Order:       test,       /*list all components in the
                           order they are to be bound*/
             test_utility,
             test_init,
             reset;

Addname:     test,
             test_utility, /*add so that link can be
                             snapped to version in
                             bound segment*/
             reset;

No_Table;                /*omit all symbol tables*/

objectname:  reset;
retain:      reset;

objectname:  test;
retain:      test;

objectname:  test_utility;
synonym:     rest_of_test; /*another entrypoint*/
no_link:     test_utility; /*do not prelink to this
                             entrypoint; generate
                             external link*/
             table;        /*keep this component's
                             symbol table*/
```

bool

SYNTAX AS A COMMAND:

```
bool B1 B2 B3
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[bool B1 B2 B3]
```

FUNCTION: performs bit string operations on character string representations of bit strings.

ARGUMENTS:

B1, B2, and B3 are bit strings entered as 0 and 1 characters. B3 must be 4 bits long. It causes the following logical operations to be performed on B1 and B2.

B3	Name	Result
0000	clear	all zeroes
0001	and	B1 & B2
0010		B1 &^B2
0011	move B1	B1
0100		^B1 & B2
0101	move B2	B2
0110	xor	(B1&^B2) | (^B1&B2)
0111	or	B1 | B2
1000	^or	^(B1|B2) = (^B1&^B2)
1001	^xor	^((B1&^B2) | (^B1&B2)) = (^B1|B2) & (B1|^B2)
1010	invert B2	^B2
1011		^(^B1&B2) = (B1|^B2)
1100	invert B1	^B1
1101		^(B1&^B2) = (^B1|B2)
1110	^and	^(B1&B2) = (^B1|^B2)
1111	^clear	all ones

NOTES: The shorter of the two strings is extended at the right with zeroes to equal the length of the longer string.

bool

EXAMPLES:

```
! string [bool 1010 0101 0111]
  1111
! string [bool 1001001 1101001010 0110]
  0100000010
```

branches

SYNTAX AS A COMMAND:

```
branches star_names {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[branches star_names {-control_arg}]
```

FUNCTION: returns the entrynames or absolute pathnames of segments, directories, and multisegment files that match one or more star names.

ARGUMENTS:

star_name
is a star name to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames. The default is to return entrynames.

NOTES: Only one name per branch is returned; i.e., if a branch has more than one name that matches star_name, only the first match found is returned.

Since each entryname (or pathname) returned by branches is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

A synonym for branches is nonlinks.

branches

EXAMPLES:

```
!  pwd
   >udd>Apple>Jones
!  ls -a

   Segments = 7, Lengths = 6.

   r w    0  empty_seg
   re     1  test
   r w    1  test.list
   r w    1  test.pl1
   re     1  prog
   r w    1  prog.list
   r w    1  prog.pl1

   Multisegment-files = 2, Lengths = 770.

   r w  513  prog.output
   r w  257  prog.data

   Directories = 2.

   sma  prog_stuff
   sma  documents

   Links = 3.

   prog.temp2                    >udd>Apple>Jones>temp_seg_2
   prog.temp1                    >udd>Apple>Jones>temp_seg_1
   junk                          >udd>Apple>Jones>empty_seg

!  string [branches prog*.**]
   prog.pl1 prog.list prog.data prog_stuff prog prog.output
!  string [branches prog.*]
   prog.pl1 prog.list prog.data prog.output
```

calc

SYNTAX AS A COMMAND:

calc {expression}

SYNTAX AS AN ACTIVE FUNCTION:

[calc expression]

FUNCTION: provides the user with a calculator capable of evaluating arithmetic expressions with operator precedence, a set of often-used functions, and a memory that is symbolically addressable (i.e., by identifier).

ARGUMENTS:

expression
is an arithmetic expression (see below) to be evaluated. If this argument is specified, the calc command prints its value and returns to command level. The expression arguments must be quoted if it contains spaces or other command language characters. Variables are not allowed.

NOTES: Invocation of calc with a newline enters calculator mode. The user can then type in expressions, assignment statements, or list requests, separated from each other by one or more newline characters. All of these operations are described below.

The user must use the quit request with a newline character to return to command level.

NOTES ON EXPRESSIONS: Arithmetic expressions involving real values and the operands +, -, *, /, and ** (addition, subtraction, multiplication, division, and exponentiation) can be typed in. A prefix of either plus or minus is allowed. Parentheses can be used, and blanks between operators and values are ignored. Calc evaluates each expression according to rules of precedence and prints out the result. The quit request (followed by a newline character) returns the user to command level. The order of evaluation is as follows--

expressions within parentheses

function references

prefix +, prefix -

**

*, /

+, -

For example, if the user types--

```
2 + 3 * 4
```

calc responds--

```
=   14
```

Operations of the same level are processed from left to right except for the prefix plus and minus, which are processed from right to left. This means 2**3**4 is evaluated as (2**3)**4.

Numbers can be integers (123), fixed point (1.23) and floating point (1.23e+2, 1.23e2, 1.23E2, or 1230E-1). All are stored as float bin(27). An accuracy of about seven figures is maintained. Variables (see below) can be used in place of constants, e.g., pi * r ** 2.

Seven functions are provided: sin, cos, tan, atan, abs, ln, and log (ln is base e, log is base 10). They can be nested to any level, e.g., sin(ln(var).5*pi/180).

calc

NOTES ON ASSIGNMENT STATEMENTS: The value of an expression can be assigned to a variable. The name of the variable must be from one to eight characters in length and must be made up of letters (uppercase and/or lowercase) and the underscore character (_). The form is--

<variable>=<expression>

For example, the following are legal assignment statements--

```
x = 35

Rho = sin(2*theta)
```

The calc command does not print any response to assignment statements. The variables "pi" and "e" have preassigned values of 3.14159265 and 2.7182818, respectively.

NOTES ON THE LIST REQUEST: If "list" is typed, calc prints out the names and values of all the variables that have been declared so far. The value of any individual variable can be displayed by typing the name of the variable followed by a newline character.

EXAMPLES: The lines typed by the user are preceded by an exclamation mark (!).

```
! calc
! 2+2
   =   4
! r = 1.5
! pi*r**2
   =   7.068583
! sin(0.01)
   =   9.999832E-3
! 143e11+(12e13
   too few )
! 143e11+(12e13)
   =   1.343E+14
! list

   r      =   1.5
   e      =   2.718282
   pi     =   3.141592
! q
```

SYNTAX AS A COMMAND:

calendar {DT} {paths} {-control_arg}

FUNCTION: prints a calendar page for one month.

ARGUMENTS:

DT
identifies which month is printed. This argument must be a date acceptable to the convert_date_to_binary_ subroutine (described in the MPM Subroutines). If the DT argument is not given, the current month is printed.

pathi
is the pathname of a segment that contains a list of events in the form of text to be inserted into the calendar. For information on segment format, see "Notes" below.

CONTROL ARGUMENTS:

-fw
labels boxes with fiscal week numbers. The calendar command assumes that each fiscal week begins on Monday and ends on Sunday and fiscal week 1 is the first full week of the calendar year. Fiscal week 1 of 1980 therefore begins on Monday, January 7, 1980.

NOTES: Each box for a calendar day is 16 characters wide and 7 high. Each box in the calendar contains the number of the day of the month; other information can also appear in the box, at the user's option. The month preceding the specified month and the month following it are also printed.

Each segment contains lines that set up a string to be inserted into the appropriate box of the calendar. The fields in these lines are separated by commas and have the form--

date,07/04,Independence Day

The first field is the operation code (either date, rel, easter or rename). The second and succeeding fields depend on which operation code is used. Lines that produce a date not in the current month are ignored. Lines beginning with an asterisk (*) are comment lines. Leading space is NOT allowed.

calendar

There are three fields for the date operation code with the first field containing the operation code, date. The second field is any date acceptable to the convert_date_to_binary_ subroutine. (This date is converted relative to the day before the beginning of the month, so that "Mon" is the first Monday in the month, etc.) The third field is arbitrary text. Up to 16 characters are inserted into the calendar in the appropriate place, if the date specified in the line (see example above) falls in the calendar month.

There are five fields for the rel operation code. The first contains the operation code, rel, itself; the second is the one or two digit month number, or 0, -1, or +1. A month of 0 is the current month, -1 is the month preceding the current month, and +1 is the month following the current month. The third is a date, relative to the day before the first of the previous month. The fourth field is a date relative to the third field and specifies the day selected. The fifth field is text. For more complete information on how these can be specified, see the convert_date_to_binary_$relative subroutine entrypoint in the MPM Subroutines. Thus, the line--

```
rel,11,Mon,Tue,Election Day
```

defines the first Tuesday after the first Monday in November, and places the text, "Election Day," in the proper calendar day box.

There are only two fields for the easter operation code. The second is the text (e.g., "Easter") that is inserted into the box for Easter.

```
easter,Easter
```

The fourth operation code is rename which has three fields; the first being the name of the operation code, the second is an existing day or month name, and the third field is the character string to replace it.

```
rename, Monday, Lundi
```

Users can insert up to six lines of text for any date. This is accomplished by supplying multiple date or rel entries for the desired date (see Washington's birthday under "Examples" below).

EXAMPLES: The following illustrates the kind of segment a user might create to put fixed holidays into a calendar.

```
* holidays
*
date,01/01,New Year's Day
date,02/02,Ground Hog Day
rel,2,Mon,2 weeks,Washington's
rel,2,Mon,2 weeks, birthday
easter,Easter
rel,4,Mon,2 weeks,Patriot's Day
rel,5,Sun,1 week,Mother's Day
rel,5,05/24,Mon,Memorial Day
date,07/04,Independence Day
rel,9,0,Mon,Labor Day
rel,10,Mon,1 week,Columbus Day
date,11/11,Veterans Day
rel,11,Mon,Tue,Election Day
rel,11,Thu,3 weeks,Thanksgiving
date,12/25,Christmas Day
```

Additionally, a user might create a segment to include personal information in a calendar.

```
* personal calendar info
*
date,12/10,Mike's Birthday
date,03/07,Dad's Birthday
   .
   .
   .
   .
```

Assume that the user wants a calendar for the coming December, including fiscal week numbers, holidays, and personal information. If the above segments are named "holidays" and "personal" (and are in the working directory), the user types the following to print the calendar on the terminal:

```
calendar 12/01 -fw holidays personal
```

cancel_abs_request (car)

SYNTAX AS A COMMAND:

car request_identifiers {-control_args}

FUNCTION: allows a user to delete a request for an absentee computation that is no longer needed.

ARGUMENTS:

request_identifiers
can be chosen from the following--

path
is the full or relative pathname for the absentee input segment of the request to be cancelled. The star convention can be used to match the entry names of segments.

-entry STR, -et STR
identifies the request to be cancelled by STR, the entryname portion of the absentee input segment pathname. The star convention is allowed.

-id ID
identifies the request to be cancelled by its request ID number. See the MPM Reference Guide for a description of request ID's.

CONTROL ARGUMENTS:

-foreground, -fg
specifies that the foreground absentee queue contains the request(s) to be cancelled.

-queue N, -q N
specifies that absentee queue N contains the request to be cancelled, where N is an integer specifying the number of the queue. The default queue is 3. For convenience in writing exec_coms and abbreviations, the word foreground or fg following the -queue control argument performs the same function as the -foreground control argument. If the -queue, -fg, and -all control arguments are omitted, only the default priority queue is searched.

-all, -a
indicates that all priority queues are to be searched starting with the highest priority queue and ending with the lowest priority queue.

cancel_abs_request (car)

-brief, -bf
suppresses messages telling that a particular request identifier was not found or that requests were cancelled when using star names or the -all control argument.

-sender STR
specifies that only requests from sender STR should be cancelled. One or more request identifiers must also be specified. In most cases, the sender is an RJE station identifier.

-user User_id
specifies the name of the submitter of the request to be cancelled, if it is not the same as the group identifier of the process. The User_id can be specified as Person_id.Project_id, Person_id, or .Project_id. This control argument is primarily for operators and administrators. Both r and d extended access to the queue are required.

ACCESS REQUIRED: The user must have read (r) and delete (d) extended access to the queue.

NOTES: The -queue, -foreground, and -all control arguments are mutually incompatible.

Normally, deletion can be made only by the user who originated the request.

When star names are not used and a single request identifier matches more than one request in the queue(s) searched, none of the requests are cancelled. However, a message is printed telling how many matching requests there are.

If the absentee process has already logged in, the user is given the choice of bumping the job and cancelling the request from the queue, or allowing the job to continue running and remain in the queue. This allows the user to cancel a running absentee process.

cancel_abs_request (car)

EXAMPLES:

The command line:

```
! car >udd>Demo>Jones>dump>translate
```

deletes the absentee request that the user had made in queue 3 that was associated with the control segment >udd>Demo>Jones>dump>translate.absin.

The command line:

```
! car >udd>Demo>Jones>doc>**.draft
```

deletes the absentee requests that the user made in queue 3 that were associated with all control segments ending with the ".draft.absin" component combination found in the >udd>Demo>Jones>doc directory.

cancel_cobol_program (ccp)

SYNTAX AS A COMMAND:

ccp names {-control_arg}

FUNCTION: causes one or more programs in the current COBOL run unit to be cancelled.

ARGUMENTS:

names
are the reference names of COBOL programs that are active in the current run unit. If the name specified in the PROG-ID statement of the program is different from its associated namei argument, namei must be in the form refname$PROG-ID.

CONTROL ARGUMENTS:

-retain_data, -retd
leaves the data segment associated with the program intact for debugging purposes. (See "Notes" below.)

NOTES: The results of the cancel_cobol_program command and the execution of the CANCEL statement from within a COBOL program are similar. The only difference is that if a namei argument is not actually a component of the current run unit, an error message is issued and no action is taken; for the CANCEL statement, no warning is given in such a case.

To preserve program data for debugging purposes, the -retain_data control argument should be used. The data associated with the cancelled program is in its last used state; it is not restored to its initial state until the next time the program is invoked in the run unit.

Cancelling ensures that the next time the program is invoked within the run unit, its data is in its initial state. Any files that have been opened by the program and are still open are closed and the COBOL data segment is truncated.

Refer to the run_cobol command for information concerning the run unit and the COBOL runtime environment. Also refer to the related commands display_cobol_run_unit (dcr) and stop_cobol_run (scr).

cancel_daemon_request (cdr)

SYNTAX AS A COMMAND:

cdr request_identifiers {-control_args}

FUNCTION: deletes an I/O daemon request that is no longer needed.

ARGUMENTS:
request_identifiers can be chosen from the following:

path
is the full or relative pathname of the input segment. The star convention is allowed to match the entrynames of segments.

-entry STR, -et STR
identifies the request to be cancelled by STR, the entryname portion of the input segment pathname. The star convention is allowed.

-id ID
identifies the request to be cancelled by its request ID number. See the MPM Reference Guide for a description of Request ID's.

CONTROL ARGUMENTS:

-request_type STR, -rqt STR
indicates that the request to be cancelled is to be found in the queue for the request type identified by the string STR. If this control argument is not given, the default request type is "printer". Request types can be listed by the print_request_types command.

-queue N, -q N
specifies that queue N of the request type contains the request to be cancelled, where N is a decimal integer specifying the number of the queue. If this control argument is omitted, only the default queue for the request type is searched. This control argument is incompatible with the -all control argument.

cancel_daemon_request (cdr)

-all, -a
searches all priority queues for the specified request type starting with the highest priority queue and ending with the lowest priority queue. This control argument is incompatible with the -queue control argument.

-brief, -bf
suppresses messages telling that a particular request identifier was not found or that requests were cancelled when using star names or the -all control argument.

-user User_id
specifies the name of the submitter of the request to be cancelled, if not the group identifier of the process. The User_id can be equal to Person_id.Project_id, Person_id, or .Project_id. Both r and d extended access to the queue are required. This control argument is primarily for operators and administrators.

ACCESS REQUIRED: The user must have o extended access to the queue to cancel their own requests. The user must have r and d extended access to cancel a request entered by another user.

NOTES: When star names are not used and a single request identifier matches more than one request in the queue(s) searched, none of the requests are cancelled. However, a message is printed telling how many matching requests there are.

Normally, deletion can be made only by the user who originated the request.

See the descriptions of the dprint and dpunch commands in this manual.

cancel_daemon_request (cdr)

EXAMPLES: The command line:

```
! cdr >udd>Alpha>Jones>dump>translate.list
```

deletes the request that the user made in queue 3 of the default request type printer, to print the segment >udd>Alpha>Jones>dump>translate.list.

The command line:

```
! cdr >udd>Alpha>Jones>dump>probe.pl1 -request_type punch
```

deletes the request that the user made in queue 3 of request type "punch" to punch the segment >udd>Alpha>Jones>dump>probe.pl1.

The command line:

```
! cdr joe sam *.*
```

cancels the requests to print segments joe, sam, and any requested segments with two-component entrynames in the current working directory in queue 3 of the default request type.

SYNTAX AS A COMMAND:

cnr -id reservation_id {-control_arg}

FUNCTION: cancels reservations made with the reserve command using the reservation identifier obtainable from the list_resources command.

ARGUMENTS:

reservation_id
must be present and is the reservation identifier of the reservation to be cancelled. It must be preceded by the -id control argument.

CONTROL ARGUMENTS:

-priv
allows a privileged cancellation to be done, such as the cancellation of a reservation belonging to another user. Use of -priv requires access to rcp_sys_.

NOTES: Reservation identifiers can be obtained by using the list_resources command.

cancel_retrieval_request (crr)

SYNTAX AS A COMMAND:

crr request_identifiers {-control_args}

FUNCTION: allows a user to delete a request for a volume retrieval that is no longer needed.

ARGUMENTS:
request_identifiers can be chosen from the following:

path
is the full or relative pathname of the segment or subtree of the retrieval request to be cancelled. The star convention is allowed to match the entrynames.

-entry STR, -et STR
identifies the request to be cancelled by STR, the entryname portion of the segment or subtree pathname. The star convention is allowed.

-id ID
identifies the request to be cancelled specified by its request ID number. See the MPM Reference Guide for a description of request ID's.

CONTROL ARGUMENTS:

-queue N, -q N
specifies that retrieval queue N contains the request to be cancelled, where N is a decimal integer specifying the number of the queue. If this control argument is omitted, only the default priority queue is searched. This control argument is incompatible with the -all control argument.

-all, -a
indicates that all retrieval queues are to be searched starting with the highest priority queue and ending with the lowest priority queue. This control argument is incompatible with the -queue control argument.

-brief, -bf
suppresses messages telling the user that a particular request identifier was not found or that requests were cancelled when using star names or the -all control argument.

cancel_retrieval_request (crr)

-user User_id
specifies the name of the submitter of the requests to be cancelled, if not equal to the group identifier of the process. The User_id can be Person_id.Project_id, Person_id, or .Project_id. Both r and d extended access to the queue are required. This control argument is primarily for operators and administrators.

ACCESS REQUIRED: The user must have read (r) and delete (d) extended access to the queue.

NOTES: Normally, deletion can be made only by the user who originated the request.

When star names are not used and a single request identifier matches more than one request in the queue(s) searched, none of the requests are cancelled. However, a message is printed telling how many matching requests there are.

EXAMPLES:

The command line:

```
! crr >udd>Demo>Jones>dump>translate
```

deletes the retrieval request for the specified segment or subtree that the user had made in queue 3.

canonicalize (canon)

SYNTAX AS A COMMAND:

canon path1 {path2} {-control_arg}

FUNCTION: ensures that the contents of a segment are in canonical form.

ARGUMENTS:

path1
is the pathname of the input segment.

path2
is the pathname of the output segment. If path2 is omitted, path1 is overwritten with the canonicalized contents of the input segment.

CONTROL ARGUMENTS:
can be -tabs in one of the following two forms:

-tabs -every X
inserts tabs at 1+n*X (where n= 1, 2, 3, ...).

-tabs n1,n2,...,n20
inserts tabs at the tab stops specified. Up to 20 tab stops can be given. No spaces are allowed in the list.

NOTES: The command ensures that all characters in a print position are sorted in the proper order and removes all ASCII carriage return (015) characters. When the -tabs control argument is specified, the canonicalize command replaces blank spaces with the appropriate tab stops. Conversely, if the -tabs argument is not specified, horizontal tab stops are replaced by the correct number of blank spaces.

EXAMPLES: To canonicalize the segment named my_prog and establish tab stops at three specified positions, the user might type:

```
! canon my_prog -tabs 7,21,35
```

To canonicalize the same segment, rename it to new_prog, and set up tab stops at 10-space intervals, the user might type:

```
! canon my_prog new_prog -tabs -every 10
```

To canonicalize the segment old_prog, which already contains tab stops that are now to be replaced with blank spaces, the user can accomplish both operations in one pass by typing:

```
! canon old_prog
```

ceil

SYNTAX AS A COMMAND:

```
ceil num
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[ceil num]
```

FUNCTION: returns the smallest decimal integer greater than or equal to its argument.

EXAMPLES:

```
! string [ceil 4.7]
5
! string [ceil -3.5]
-3
```

change_default_wdir (cdwd)

SYNTAX AS A COMMAND:

cdwd {path}

FUNCTION: records a specified directory as the user's default working directory for the duration of the current process or until the next change_default_wdir command is issued.

ARGUMENTS:

path
is the pathname of a directory. If path is not specified, the current working directory becomes the default working directory.

NOTES: The change_default_wdir command is used in conjunction with the change_wdir command. When the change_wdir command is issued with no argument, the default working directory becomes the current working directory.

The original default working directory is the user's home directory upon logging in.

See also the descriptions of the change_wdir (cwd) and print_default_wdir (pdwd) commands.

change_error_mode (cem)

SYNTAX AS A COMMAND:

cem {-control_args}

FUNCTION: controls the amount of information printed by the default handler for system conditions. It determines the length of messages for the life of a process or until it is invoked again in the process.

CONTROL ARGUMENTS:

-brief, -bf
prints only the condition name.

-long, -lg
prints more complete messages. In particular, if the condition was detected in a support procedure, the name of that procedure is printed in addition to the name of the most recent user procedure. If a segment that signalled a condition (or caused it to be signalled) is bound, both the offset relative to the base of the procedure and the offset relative to the base of the segment are printed.

NOTES: If this command is not issued or is issued with no control arguments, the user receives default length error messages. Default length messages are intermediate in length between the brief and long messages.

For a complete discussion of conditions and their handling see the MPM Reference Guide. Refer to the description of the reprint_error command for a similar, but more selective, capability.

change_wdir (cwd)

SYNTAX AS A COMMAND:

cwd {path}

FUNCTION: changes the user's working directory to the directory specified as an argument.

ARGUMENTS:

path
is the pathname of a directory. If path is not specified, the default working directory is assumed.

ACCESS REQUIRED: The user must have s permission on the directory containing path, but no access to path is required.

NOTES: A working directory is a directory in which the user's activity is centered. Its pathname is remembered by the system so that the user need not type the full absolute pathname of segments inferior to that directory.

If path specifies a nonexistent directory, an error message is printed on the user's terminal and the current working directory is not changed.

No access to path is required for this command to be employed. However, once the working directory has been changed, the user can proceed only according to the user's access to path. That is, to effectively use path as a working directory, the user must have sma access permission for path; however, restricted uses are possible in accordance with the access mode attributes on the directory. For example, the user must have at least status permission to list the directory.

See also the descriptions of the change_default_wdir (cdwd) and print_default_wdir (pdwd) commands.

check_iacl

SYNTAX AS A COMMAND:

```
check_iacl {path} {-control_args}
```

FUNCTION: lists segments whose access control lists (ACLs) disagree with the initial ACL for segments.

ARGUMENTS:

path
is the pathname of the directory whose segment ACLs are to be checked against the segment initial ACL. If path is omitted, the working directory is assumed.

CONTROL ARGUMENTS:

-all, -a
lists User_ids in a segment ACL but not contained in the initial ACL; also lists User_ids included in the initial ACL but omitted from a segment ACL. If this control argument is not specified, only User_ids in addition to those in the initial ACL are listed.

-exclude User_id, -ex User_id
excludes the specified User_id from the comparison. Up to ten -exclude control arguments can be specified. The star convention is allowed.

EXAMPLES:

```
! check_iacl

   oldMap.com.runoff
     ACL added: rew Jones.Demo.*
     ACL added: rew Jordan.Work.*

   add_search.com.runoff
     ACL added:  rew Jones.Demo.*
```

check_info_segs (cis)

SYNTAX AS A COMMAND:

cis {-control_args}

FUNCTION: prints a list of new or modified info segments. It saves the current time in the user profile, so that when it is invoked again, it lists segments created or modified since the last invocation.

CONTROL ARGUMENTS:

-date DT, -dt DT
If this argument is specified, check_info_segs uses the date specified by DT instead of the date in the user profile. The DT argument must be acceptable to the convert_date_to_binary_ subroutine (described in the MPM Subroutines). The time of last invocation in the user profile is not updated to the current time.

-long, -lg
If this argument is specified, check_info_segs lists the date-time-entry-modified as well as the name of any segment selected as having been created or modified during the interval in question.

-brief, -bf
If this argument is specified, check_info_segs does not print the names of selected segments and suppresses the comment "no change" if no segments are selected as having been created or modified during the interval in question. This control argument is intended for use with the -call control argument described below.

-no_update, -nud
If this argument is specified, check_info_segs does not place the current time into the user profile.

-call cmdline
If this argument is specified, check_info_segs calls the command processor with a string of the form "cmdline path" for each selected segment, after the name of the segment is typed; path is the absolute pathname of the segment. The cmdline must be enclosed in quotes if it contains blanks.

-pathname spath, -pn spath
If this control argument is specified, check_info_segs assumes that spath is a pathname with one or more asterisks (stars) in the entryname portion. All new or modified segments that

check_info_segs (cis)

match spath are selected. Refer to "Constructing and Interpreting Names" in the MPM Reference Guide for a discussion of star names.

All specified directories are searched, in the order that the arguments are given. If the -pathname argument is not specified, the default is to search the directories in the "info_segments" search list for "**.info". See "Notes on Search List" section below.

NOTES: The first time check_info_segs is invoked by a particular user, it just initializes the time in the user profile to the current time, prints a comment, and does not list any segments. If a profile does not exist, check_info_segs creates one in the user's home directory. The profile segment has the name Person_id.profile, where Person_id is the Person_id given at login time.

The check_info_segs command checks the date-time-entry-modified for any segment pointed to by a link, not the time the link was modified.

The check_info_segs command cannot detect that a segment has been deleted since the last invocation of the command.

NOTES ON SEARCH LIST:

The check_info_segs command uses the "info_segments" search list which has the synonyms "info_segs" and "info". The default "info_segments" search list is:

```
>doc>iml_info
>doc>info
```

These directories contain info segments provided by the site and those supplied with the system. Type "psp info" to see what the current "info" search list is. For more information about search lists, see the search facility commands, and in particular, the add_search_paths description in this manual.

EXAMPLES: To check for info segments modified since the specified date, type:

```
! cis -date "07/01/79 0900."
```

check_info_segs (cis)

To print all modified info segments, type:

```
! cis -call print -brief
```

The -brief control argument is given to check_info_segs to suppress duplicate printing of segment names since the print command types the segment name in the heading.

To print just the first block of any modified info segment, type:

```
! cis -call "answer no help"
```

To check for all modified segments in a project-maintained directory >udd>Project_id>doc as well as the default directories, type the following two command lines:

```
! asp info_segments >udd>Project_id>doc
! cis
```

close_file (cf)

SYNTAX AS A COMMAND:

cf {-control_arg} filenames

FUNCTION: closes specified FORTRAN and PL/I files. It closes all open FORTRAN and PL/I files if the -all control argument is specified.

ARGUMENTS:

filenames
are the names of open FORTRAN or PL/I files.

CONTROL ARGUMENTS:

-all, -a
closes all open files. In this case, no filename appears.

NOTES: The format of a FORTRAN file name is filenn where nn is a two-digit number other than 00; e.g., file05. PL/I file names are selected by the user and can have any format.

If a specified file cannot be found, an error message is printed indicating the name of the file. The rest of the specified files are closed.

For each filename, all PL/I files of that name and, if applicable, the FORTRAN file of that name are closed.

The command "close_file -all" does not affect I/O switches that are not associated with FORTRAN or PL/I files.

SYNTAX AS A COMMAND:

cobol path {-control_args}

FUNCTION: invokes the COBOL compiler to translate a segment containing the text of a COBOL source program into a Multics object segment.

ARGUMENTS:

path
is the pathname of a COBOL source segment to be translated by the COBOL compiler. If path does not have a suffix of cobol, one is assumed. However, the suffix cobol must be the last component of the name of the source segment. If the expand_cobol_source command is used to create a new segment with the suffix ex.cobol, a check is made to see if this segment exists and it is used.

CONTROL ARGUMENTS:

-brief, -bf
causes error messages written to the user_output I/O switch to contain only an error number and statement identification, once the full message has been given on the first occurrence. In the normal, nonbrief mode, an explanatory message is printed for each occurrence.

-check, -ck
is used for syntactic and semantic checking of a COBOL program. No code is generated.

-expand, -exp
accepts a source segment in the format acceptable to the expand_cobol_source command. It expands the source segment by evaluating COPY and REPLACE statements. If the segment to be translated has the suffix ex.cobol, this control argument is ignored.

-format, -fmt
accepts a source segment in the format acceptable to the expand_cobol_source command. If the segment to be translated has the suffix ex.cobol, this control argument is ignored.

-levelN, -levN
causes severity three L-type diagnostics to be written to the user_output I/O switch whenever a COBOL source line contains a language construct outside the subset specified by N. The

cobol

value N can be one through five, corresponding to the four levels specified by the Federal Information Processing Standards Publication, December 1, 1975 (FIPS PUB 21-1) and to the extended version of COBOL supported by Multics. These values are:

1 low level
2 low intermediate level
3 high intermediate level
4 high level
5 Multics COBOL extensions

If a program compiles without any L-type diagnostics, it means the program is an acceptable subset of Multics COBOL at the level requested. The default is level 5.

-list, -ls
produces a source program listing with symbols, followed by an assembly-like listing of the compiled object program. Use of the -list control argument significantly increases compilation time and should be avoided whenever possible by using the -map control argument.

-map
produces a source program listing with symbols, followed by a map of the object code generated by this compilation. The -map control argument produces sufficient information to allow the user to debug most problems online.

-profile, -pf
generates additional code to meter the execution of individual statements. Each statement in the object program contains an additional instruction to increment an internal counter associated with that statement. After a program has been executed, the profile command can be used to print the execution counts.

-runtime_check, -rck
produces an object program in which parameters are validated according to number and type, performs bounds checking on all subscripted referenced, performs string range checking on all variable length string references, and verifies the validity of every index name modification.

-severityN, -svN
causes error messages whose severity is less than N (where N is 1, 2, 3, or 4) to not be written to the user_output I/O switch. All errors are written into the listing. If this control argument is not given, a severity level of 2 is

assumed. See the description of severity levels under "Notes on Error Diagnostics" below.

-table, -tb
generates a full symbol table for use by symbolic debuggers. The symbol table is part of the symbol section of the object program and consists of two parts: a statement table that gives the correspondence between source line numbers and object locations and an identifier table that contains information about every identifier actually referenced by the source program. The table appears in the symbol section of the object segment produced by the compilation. This control argument usually causes the object segment to become significantly longer. If the -format control argument is given with the -table control argument, the symbolic debuggers are not able to display the source statements.

-temp_dir path, -td path
creates the compiler's internal work files in the specified directory rather than in the process directory. This control argument may be necessary for very large source files (over approximately 3000 lines) that incur record quota overflow in the process directory during compilation.

-debug, -db
leaves the work files generated by the compiler intact after a compilation. This control argument is used for debugging the compiler. The command cobol$clean_up can be used to discard these files. Also, this causes severity 4 errors to not unwind and abort the compilation, but rather to invoke a new level of the command processor at the point of the error.

-time, -tm
prints the time (in seconds) and the number of page faults taken by each phase of the compiler; prints the total time at the end of the compilation. This information is directed to the user_output I/O switch.

NOTES: The only result of invoking the cobol command without control arguments is to generate an object segment.

A normal compilation produces an object segment and leaves it in the user's working directory. If an entry with that name already exists in the directory, its access control list (ACL) is saved and given to the new copy of the object segment. Otherwise, the user is given re access to the segment with ring brackets v,v,v where v is the validation level of the process that is active when the object segment is created.

If the user specifies the -map or -list control arguments, the cobol command creates a listing segment in the working directory and gives it a name consisting of the entryname portion of the source segment with a suffix of list rather than cobol (e.g., a source segment named business.cobol would have a listing segment named business.list). The ACL is set as described for the object segment except that the user is given rw access to it when newly created. Previous copies of the object segment and the listing segment are replaced by the new segments created by the compilation.

A listing segment can also be produced. These segments are placed in the user's working directory.

This command cannot be called recursively.

For information on COBOL, refer to the Multics COBOL Users' Guide, Order No. AS43 and the Multics COBOL Reference Manual, Order No. AS44. See the description of the profile command in this document.

NOTES ON ERROR DIAGNOSTICS: The COBOL compiler can diagnose and issue messages for about 800 different errors. These messages are graded in severity as follows:

1 Warning only. Compilation continues without ill effect.

2 Correctable error. The compiler attempts to remedy the situation and continues, possibly without ill effect. The assumptions the compiler makes in remedying the situation, however, do not necessarily guarantee the right results.

3 Uncorrectable but recoverable error. That is, the program is definitely in error and no meaningful object code can be produced, but the compiler can continue executing and diagnosing further errors.

4 Unrecoverable error. The compiler cannot continue beyond this error. A message is printed and control is returned to the cobol command. The command writes an abort message on the error_output I/O switch and returns to its caller.

As indicated above, the user can set the severity level so as not to be bothered by minor error messages. The user can also specify the -brief control argument so that the message is shorter. Since the default severity level is 2, the user must explicitly specify the -severity1 (or -sv1) control argument when invoking the cobol command to have warning messages printed. Neither the -severityN nor -brief control argument has any effect on the contents of the listing segment if one is produced.

An example of an error message in its long form is:

```
22          use after error procedure on extend.
                                                 1
** 1 5-250 A use procedure has already been associated with
this processing mode.
```

If the -brief control argument is specified and message 5-250 has previously been given in its long form, the user instead sees:

```
22          use after error procedure on extend.
                                                 1
** 1 5-250
```

If the user has set the severity level to 3, no message is printed at all. Notice that the number of asterisks immediately preceding the error indicator corresponds to the severity level of the error.

If a listing is produced, the error messages appear interspersed with the lines of the source program. No more than 300 messages are printed in the listing.

NOTES ON LISTING: The listing created by the cobol command is a line-numbered image of the source segment with diagnostics interspersed. This is followed by a cross-reference table of all the names defined within the program. Following the cross-reference table is the object code map, which gives the starting location in the text segment of the instructions for each statement in the program. The map is sorted by ascending storage locations. Finally, the listing contains an assembly-like list of the object code produced. The executable instructions are grouped under an identifying header, which contains the source statement that produced the

cobol

instruction. Opcode, pointer-register, and modifier mnemonics are printed alongside the octal instruction. If the address field of the instruction uses the IC (self-relative) modifier, the absolute text location corresponding to the relative address is printed on the remarks field of the line.

cobol_abs (cba)

SYNTAX AS A COMMAND:

cba paths {cobol_args} {dp_args} {abs_control_args}

FUNCTION: submits an absentee request to perform COBOL compilations. The absentee process for which cobol_abs submits a request compiles the segments named and prints and deletes the listing segment.

ARGUMENTS:

paths
are the pathnames of segments to be compiled.

cobol_args
can be one or more control arguments accepted by the cobol command.

dp_args
can be one or more control arguments (except -delete, -dl) accepted by the dprint command.

LIST OF ABSENTEE CONTROL ARGUMENTS:
abs_control_args can be chosen from the following:

-queue N, -q N
specifies in which priority queue the request is to be placed (N < 3). The default queue is 3; the listing segment is printed in dprint queue N.

-hold, -hd
specifies that cobol_abs should not print or delete the listing segment.

-limit N, -li N
places a limit on the CPU time used by the absentee process. The parameter N must be a positive decimal integer specifying the limit in seconds. The default limit is defined by the site for each queue. An upper limit is defined by the site for each queue on each shift. Jobs with limits exceeding the upper limit for the current shift are deferred to a shift with a higher limit.

-output_file path, -of path
specifies that absentee output is to go to the segment whose pathname is path.

cobol_abs (cba)

NOTES: Control arguments and segment pathnames can be mixed freely and can appear anywhere on the command line after the command. All control arguments apply to all segment pathnames. If an unrecognizable control argument is given, the absentee request is not submitted.

Unpredictable results can occur if two absentee requests are submitted that could simultaneously attempt to compile the same segment or write into the same absout segment.

When doing several compilations, it is more efficient to give several segment pathnames in one command rather than several commands. With one command, only one process is set up. Thus the dynamic intersegment links that need to be snapped when setting up a process and when invoking the compiler need be snapped only once.

If the -output_file control argument is not specified, an output segment, path.absout, is created in the user's working directory (if more than one path is specified, only the first is used).

If none of the segments to be compiled can be found, no absentee request is submitted.

collate

SYNTAX AS A COMMAND:

collate

SYNTAX AS AN ACTIVE FUNCTION:

[collate]

FUNCTION: returns the 128 characters of the ASCII character set in collating sequence.

collate9

SYNTAX AS A COMMAND:

collate9

SYNTAX AS AN ACTIVE FUNCTION:

[collate9]

FUNCTION: returns a character string containing all possible 9-bit bit patterns rather than just the 128 ASCII characters, therefore, making the returned string 512 characters long.

compare

SYNTAX AS A COMMAND:

compare path1{|offset1} path2{|offset2} {-control_args}

FUNCTION: compares two segments and lists their differences. The comparison is a word-by-word check and can be made with a mask so that only specified parts of each word are compared.

ARGUMENTS:

path1, path2
are the pathnames of the segments to be compared. The equal convention is allowed for path2.

offset1, offset2
are octal offsets within the segments to be compared. The comparison begins at the word specified or at the first word of the segment if no offset is specified. If an offset is omitted, the vertical bar should also be omitted.

CONTROL ARGUMENTS:

-brief, -bf
prints only the first and last words of each block of discrepancies that is four or more words in length. The default is to print all discrepancy words.

-length N, -ln N
the comparison should continue for no more than N (octal) words.

-long, -lg
prints all discrepancy words, unlike -brief. This is the default.

-mask N
the octal mask N is to be used in the comparison. If N is less than 12 octal digits, it is padded on the left with zeros.

NOTES: The maximum number of words to be compared is the word count of the first segment minus its offset or the word count of the second segment minus its offset, whichever is greater. If the -length control argument is supplied, comparison stops after the specified number of words. If the segments are of unequal length, the remaining words of the longer segment are printed as discrepancies. The word count of a segment is computed by dividing

printed as discrepancies. The word count of a segment is computed by dividing the bit count plus 35 by 36. If the word count minus the offset is less than zero, an error message is printed and the command is aborted.

Any discrepancies found by the command are listed in the following format:

offset	contents	offset	contents
4	404000000002	4	000777000023
6	404000000023	6	677774300100

To compare segments containing only ASCII character-string data, use the compare_ascii command described in this manual.

compare_ascii (cpa)

SYNTAX AS A COMMAND:

cpa paths {-control_args}

FUNCTION: compares ASCII segments and prints any differences.

ARGUMENTS:

paths
are the pathnames of the segments to be compared. Up to six segments can be compared, in addition to the original if one is supplied. The equal convention can be used in any pathname except the first one on the command line, which is assumed to be the original unless otherwise specified.

CONTROL ARGUMENTS:

-original pathA, -orig pathA
specifies the pathname pathA of the original segment of which the others are modified versions.

-no_original, -no_orig
indicates that no original segment is supplied. If neither -no_original nor -original is given, the first pathname on the command line is assumed to be the original.

-minchars NN
specifies the minimum number of characters that must be identical for compare_ascii to assume that it has found the end of a difference. The default is 20 characters. See "Notes" below.

-minlines NN
specifies the minimum number of lines that must be identical for compare_ascii to assume that it has found the end of a difference. The default is two lines. See "Notes" below.

-totals, -tt
prints only the totals line, giving the number of differences and the number of changed lines. The default is to print discrepancies and totals line.

-no_totals, -ntt
does not print the totals line.

-header, -he
prints a heading, giving the full pathname and identifying letter of each segment. This heading is not printed by

compare_ascii (cpa)

default.

-print_new_lines, -pnl
prints only new lines. New lines are lines found in one or more of the modified versions but not in the original. An original must be supplied if this argument is used.

-no_numbers, -nnb
does not print identifying letter and line numbers preceding the lines from the segments being compared. The default is to print them.

NOTES: The output is organized with the assumption that the pathA segment was edited to produce pathB. This command prints lines that were added, replaced, or deleted; it identifies each line by line number within the respective segment and also by the letter A or B to indicate which segment the line is from (A for pathA and B for pathB).

Values for minchars and minlines can be specified without being preceded by control arguments. The order is: minchars minlines.

The values of minchars and minlines control the size of displayed differences. Large values for these parameters cause small, closely-spaced differences to be displayed as one large difference, while very small values (such as -minlines 1 -minchars 2) will cause small changes to be displayed individually but might also cause large differences to be broken down into small parts, thereby giving a misleading picture of what was actually done to produce the modified versions. The user should adjust these parameters to produce the most useful results.

EXAMPLES: The examples of compare_ascii usage below are based on the segments yesterday.menu and today.menu displayed here side by side.

yesterday.menu

```
Breakfast Menu:
     Juice
     Toast
     Eggs
Luncheon Menu:
     Hot dogs
```

today.menu

```
Breakfast Menu:
     Juice
     Toast
     Eggs
Luncheon Menu:
     Hamburger
```

compare_ascii (cpa)

```
     Milk                                    Milk
     French fries                            Salad
Supper Menu:                                 French fries
     Steak                              Supper Menu:
     Baked potato                            Chicken
     Coffee                                  Rice
                                             Coffee
```

The default operation of compare_ascii is illustrated by the command line:

```
! cpa yesterday.menu today.menu

   A6          Hot dogs
   A7          Milk
   Changed by B to:
   B6          Hamburger
   B7          Milk
   B8          Salad


   A10         Steak
   A11         Baked potato
   Changed by B to:
   B11         Chicken
   B12         Rice

   Comparison finished: 2 differences, 9 lines.
```

The following command line shows the use of the -original, -header, -minlines, and -minchars control arguments. Notice that the lower values of minlines and minchars isolate the two changes within the Luncheon menu.

```
! cpa today.menu -orig yesterday.menu -he -minchars 5
     -minlines 1

   A >udd>m>Jones>yesterday.menu (original)
   B >udd>m>Jones>today.menu (new)

   A6          Hot dogs
   Changed by B to:
   B6          Hamburger


   Inserted in B:
   B8          Salad
   Preceding:
   A8          French fries
```

compare_ascii (cpa)

```
A10             Steak
A11             Baked potato
Changed by B to:
B11             Chicken
B12             Rice

Comparison finished: 3 differences, 7 lines.
```

In the following example the printing of line numbers, old lines, and the totals line have been suppressed, giving better visibility to what is new in today.menu.

```
! cpa yesterday.menu today.menu -pnl -nnb -ntt -minchars 5
      -minlines 1

Hamburger
Salad
Chicken
Rice
```

SYNTAX AS A COMMAND:

contents path

SYNTAX AS AN ACTIVE FUNCTION:

[contents path]

FUNCTION: returns the contents of a segment as a character string. Newline characters in the segment are changed to blanks in the string.

EXAMPLES: Assume that the segment named distribution contains a list of names (each person's name on a separate line).

The command line:

```
! dp -ds ([contents distribution]) output
```

prints one copy of the segment output for each name on the list, using the name as the destination.

convert_characters (cvc)

SYNTAX AS A COMMAND:

```
   cvc key1 {oldpath} {newpath}
or:
   cvc key2 char_string
```

FUNCTION: allows the rapid editing of a segment in the case where a one-for-one replacement of certain characters by certain other characters must be done. An example of its use is the conversion of all uppercase characters in a segment to lowercase characters. Keywords specify the conversion to take place. For certain of the keys convert_characters maintains a from_string and a to_string that define the conversion to be made. The converted segment is the same as the original except that every instance of the i'th character of from_string present in the original segment is replaced by the i'th character of to_string.

The conversion for the key "sp" uses a from_string and to_string that must have been previously set by use of the "from" and "to" keys.

ARGUMENTS:

key1
any of the keys listed below in "List of keywords".

oldpath
the pathname of a segment to be converted. If this argument is omitted, the from_string and to_string related to key1 are printed.

newpath
the pathname of the output segment. If this argument is omitted, newpath is assumed to be the same as oldpath, and the converted copy replaces the original.

key2
either "to" or "from" to set to_string or from_string for the "sp" key.

char_string
is the string to be set as to_string or from_string. If it contains blanks, it must be enclosed in quotes.

convert_characters (cvc)

LIST OF KEYWORDS:

lc
converts alphabetic characters to lowercase.

uc
converts alphabetic characters to uppercase.

mp
converts from Multics PL/1 format to IBM 360 PL/1.

bcd
converts BCD special characters to ASCII/EBCDIC equivalents.

dart
converts Multics special characters to corresponding Dartmouth special characters as follows:

^	'
_	=
>	"
+	<
=	>
'	:
{	+
"	?
?	

sp
uses conversion strings set earlier by the from and to keys: cvc from char_string1;cvc to char_string2

NOTES: The most recent setting of from_string and to_string in the user's process is used for conversion with the "sp" key. No conversion is attempted for the "sp" key unless both the from_string and the to_string are of the same non-zero length. Any character not present in the from_string is not changed.

SYNTAX AS A COMMAND:

cp path1_1 {path2_1 ... path1_n path2_n} {-control_args}

FUNCTION: causes copies of specified segments and multisegment files to be created in the specified directories with the specified names. Access control lists (ACLs) and multiple names are optionally copied.

ARGUMENTS:

path1_i
is the pathname of a segment or multisegment file to be copied. If path1 is the name of a link, the command copies the target of the link. The star convention is allowed.

path2_i
is the pathname of a copy to be created from path1_i. If the last path2 argument is not given, the copy is placed in the working directory with the entryname of path1_n. The equal convention is allowed.

CONTROL ARGUMENTS:

-acl
copies the ACL.

-all, -a
copies multiple names and ACLs.

-brief, -bf
suppresses the warning messages "Bit count inconsistent with current length..." and "Current length is not the same as records used...".

-chase
copies the targets of links that match path1. See "NOTES" for the default action.

-long, -lg
prints warning messages as necessary. This is the default.

-name, -nm
copies multiple names.

-no_acl
does not copy the ACL. This is the default.

-no_chase
does not copy the targets of links that match path1. See "NOTES" for the default action.

-no_name, -nnm
does not copy multiple names. This is the default.

ACCESS REQUIRED: Read access is required for path1i. Status permission is required for the directory containing path1i. Append permission is required for the directory containing path2i. Modify permission is required if the -name, -acl, or -all control argument is used.

NOTES: The control arguments can appear once anywhere in the copy command line after the command name and apply to the entire copy command line.

The default for chasing links depends on path1. If path1 is not a starname, links are chased by default. If path1 is a starname, links are not chased.

If the ACL of a segment or multisegment file is being copied, the initial ACL of the target directory has no effect on the ACL of the segment or multisegment file after it has been copied into that directory. The ACL remains exactly as it was in the original directory.

Since two entries in a directory cannot have the same entryname, special action is taken by this command if the name of the segment or multisegment file being copied (specified by path1i) already exists in the directory specified by path2i. If the entry being copied has an alternate name, the entryname that would have resulted in a duplicate name is removed and the user is informed of this action; the copying operation then takes place. If the entry being copied has only one entryname, the entry that already exists in the directory must be deleted to remove the name. The user is asked if the deletion should be done; if the user answers "no", the copying operation does not take place.

The copy command prints a warning message if the bit count of path1i is less than its current length or if the current length is greater than the number of records used. These

copy (cp)

warnings are suppressed by the use of the -brief control argument.

EXAMPLES:

The command line:

```
! copy >old_dir>fred.list george.=
```

copies segment or multisegment file named fred.list in the directory >old_dir into the working directory as george.list.

SYNTAX AS A COMMAND:

copy_acl path1*1* path2*1* {... path1*n* path2*n*}

FUNCTION: copies the access control list (ACL) from one file or directory to another, replacing the current ACL if necessary.

ARGUMENTS:

path1*i*
is the pathname of a file or directory whose ACL is to be copied. The star convention is allowed.

path2*i*
is the pathname of a file or directory onto which the initial ACL is to be copied. The equal convention is allowed.

copy_cards (ccd)

SYNTAX AS A COMMAND:

ccd deck_name {new_deck_name}

FUNCTION: copies specified card image segments from system pool storage into a user's directory. The segments to be copied must have been created using the Multics card input facility.

ARGUMENTS:

deck_name
is the name that was entered on the deck_id card when the card deck was submitted for reading. The star convention is allowed.

new_deck_name
is the pathname of the segment in which the matching card image segment is to be placed. If omitted, the working directory and deck_name are assumed. The equal convention is allowed.

NOTES: See the description of the card input facility in the MPM Reference Guide for the format of the control cards needed when submitting a card deck to be ready by system operations. The user process executing this command must have the proper access to the card image segment in order to perform the copy. When there are multiple copies of the same deck in pool storage, all are copied.

When deck_name is a starname and there are several matching card image segments in pool storage to which the user has access, all are copied.

When an attempt is made to read a card deck having the same name as some previously read deck still in pool storage, a numeric suffix is added to the name of the new deck, e.g., "deck_name.1". Repeated name duplications cause successively larger numeric suffixes to be used. (Name duplications can only occur for decks of the same access class submitted by the same user.) The copy_cards command informs the user of such duplications (if any) and retrieves all copies of the specified deck.

copy_cards (ccd)

Only those card decks having an access class equal to the user's current authorization can be copied. Other decks are not found.

EXAMPLES:

The command line:

```
! ccd my_deck
```

copies the user's card image segment named my_deck from the card pool storage into the user's current working directory.

copy_characters (cpch)

SYNTAX AS A COMMAND:

```
cpch str N
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[cpch str N]
```

FUNCTION: returns a quoted string containing N copies of a specified string.

EXAMPLES:

```
! string [cpch "1 2 3 " 3]
  1 2 3 1 2 3 1 2 3
```

copy_dir (cpd)

SYNTAX AS A COMMAND:

cpd source_dir {target_dir} {-entry_type_keys} {-control_args}

FUNCTION: copies a directory and its subtree to another point in the hierarchy.

ARGUMENTS:

source_dir
is the pathname of a directory to be copied. The star convention is allowed.

target_dir
is the pathname of the copy of the source_dir. The equal convention is allowed. If target_dir is not specified, the copy is placed in the working directory with the entryname of source_dir. If the target_dir does not exist, it is created.

entry_type_keys
control what type of storage system entries in the subtree are copied. If no entry_type_key is specified, all entries are copied. The keys are:

-branch, -br
-directory, -dr
-file, -f
-link, -lk
-multisegment_file, -msf
-non_null_link, -nnlk
-segment, -sm

If one or more entry_type_keys are specified, but not the -directory key, the subtree of source_dir is not walked.

CONTROL ARGUMENTS:

-brief, -bf
suppresses the printing of warning messages such as "Bit count is inconsistent with current length" and "Current length is not the same as records used".

-force
executes the command, when target_dir already exists, without asking the user. If the -force control argument is not specified, the user is queried.

copy_dir (cpd)

-replace, -rp
deletes the existing contents of target_dir before the copying begins. If target_dir is non-existent or empty, this control argument has no effect. The default is to append the contents of source_dir to the existing contents of target_dir.

-acl
gives the ACL on the source_dir entry to its copy in target_dir. Although initial ACLs are still copied, they are not used in setting the ACL of the new entries when this control argument is specified. See "Notes on Access Provision" below for further discussion.

-primary, -pri
copies only primary names. If the -primary control argument is not specified, all the names of the selected entries are copied.

-no_link_translation, -nlt
copies links with no change. The default is to translate links being copied. If there are references to the source directory in the link pathname of a link being copied, the link pathname is changed to refer to the target directory.

-chase
copies the target of a link. The default is not to chase links. Chasing the links eliminates link translation.

NOTES: The user can specify that portions of the subtree be copied and can control the processing of links. See also the copy, move, and move_dir commands in this manual.

ACCESS REQUIRED: Status permission is required for source_dir and all of the directories in its tree. Status permission is required for the directory containing source_dir. Read access is required on all files under source_dir. Append and modify permission are required for the directory containing target_dir if target_dir does not exist prior to the invocation of the copy_dir command. Modify and append permission are required on target_dir if it already exists. This command does not force access.

NOTES ON ACCESS PROVISION: If the -acl control argument is not specified, the system default ACLs are added, then the initial ACL for the containing directory is applied (which may change the system supplied ACL). Initial ACLs are always copied for the current ring of execution.

copy_dir (cpd)

NOTES ON THE EXISTENCE OF target_dir: If target_dir already exists and -force is not specified, the user is so informed and asked if processing should continue. If target_dir is contained in or contains source_dir, an appropriate error message is printed and control is returned to command level. Otherwise, the contents of source_dir either are appended to or replace the contents of target_dir. (See the -replace control argument.)

NOTES ON STAR AND EQUAL CONVENTIONS: The star convention in source_dir matches only directory names and copies them. Matching names associated with other storage types are ignored.

NOTES ON NAME DUPLICATIONS: Since two entries in a directory cannot have the same entry name, this command takes special action if the entryname of the entry being copied already exists in the directory specified by target_dir. If the entry is a directory, it is handled in the same fashion as duplication between source_dir and target_dir is handled, unless the existing entry in target_dir is not also a directory. In this case the entryname duplication is treated the same as non-directory entries. The procedure for non-directory entries is the standard system technique. See the copy command in this manual.

If the -replace control argument is specified or target_dir does not exist, name duplication does not occur.

NOTES ON LINK TRANSLATION: If part of the tree is not copied (by specifying a storage system entry key), problems with link translation may occur. If the link target in the source_dir tree was in the part of the tree not copied, there may be no corresponding entry in the target_dir tree. Hence, translation of the link causes the link to become null.

copy_dir (cpd)

EXAMPLE:

The command line:

```
! cpd old_source new_source -segment -acl
```

copies all the segments with their ACLs in the directory old_source to the directory new_source.

The command line:

```
! cpd old_user new_user -branch
```

copies all the segments, directories and multisegment files from the directory old_user to the directory new_user (no links are copied).

copy_file (cpf)

SYNTAX AS A COMMAND:

cpf in_control_arg out_control_arg {-control_args}

FUNCTION: copies records or lines from an input file to an output file. The copy command makes an exact duplicate of the input file, whereas copy_file produces an output file that has been restructured for maximum compactness. (See the description of the copy command in this manual.)

ARGUMENTS:

LIST OF in_control_args:
the input file from which records or lines are read can be specified by either an I/O switch name or an attach description. (See "Notes" below.)

-input_switch STR, -isw STR
specifies the input file by means of an already attached I/O switch name, where STR is the switch name.

-input_description STR, -ids STR
specifies the input file by means of an attach description STR. STR must be enclosed in quotes if it contains spaces or other command language characters.

LIST OF out_control_args:
the output file to which the records or lines are written can be specified by either an I/O switch name or an attach description. (See "Notes" below.)

-output_switch STR, -osw STR
specifies the output file by means of an already attached I/O switch name, where STR is the switch name.

-output_description STR, -ods STR
specifies the output file by means of an attach description STR. STR must be enclosed in quotes if it contains spaces or other command language characters.

CONTROL ARGUMENTS:

-keyed
copies both records and keys from a keyed sequential input file to a keyed sequential output file. The default is to

copy_file (cpf)

copy records from an input file (either keyed or not) to a sequential output file. (See "Notes on Keyed Files" below.)

-from N, -fm N
copies beginning with the Nth record or line of the input file, where N is a positive integer. The default is to begin copying with the "next record." (See "Notes" below.)

-start STR, -sr STR
copies beginning with the record whose key is STR, where STR is 256 or fewer ASCII characters. The default is to begin copying with the "next record."

-to N
copies until the Nth record or line has been copied or the input file is exhausted, whichever occurs first, where N is a positive integer greater than or equal to the N given with the -from control argument. This control argument can only be specified if -from is also specified. The default is to perform copying until the input file is exhausted.

-stop STR, -sp STR
copies until the record whose key is STR has been copied or the input file is exhausted, whichever occurs first, where STR is 256 or fewer ASCII characters. This control argument can be specified without specifying the -start control argument. However, if -start is specified, the STR given with -stop must be greater than or equal to (according to the ASCII collating sequence) the STR given with -start. The default is to perform copying until the input file is exhausted.

-count N, -ct N
copies until N records or lines have been copied or the input file is exhausted, whichever occurs first, where N is a positive integer. The default is to perform copying until the input file is exhausted.

-all, -a
copies until the input file is exhausted. This is the default.

-brief, -bf
suppresses an informative message indicating the number of records or lines actually copied.

-long, -lg
prints an informative message indicating the number of records or lines actually copied. This is the default.

-input_mode x, -imode x
specifies the opening mode for the input file. If copy_file

copy_file (cpf)

opens the file, it uses this mode. If the file is already open, this mode must be consistent with the open mode of the file (e.g., stream_input is consistent with stream_input_output). Allowable values for x are:

keyed_sequential_input, ksqi	sequential_input_output, sqio
keyed_sequential_output, ksqo	stream_input, si
sequential_input, sqi	stream_output, so
sequential_output, sqo	stream_input_output, sio

-output_mode x, -omode x
specifies the opening mode of the output file. See -input_mode above for restrictions.

-character, -ch
specifies that any positioning of stream files done by copy_file is to be done in terms of characters rather than lines.

NOTES ON UNSTRUCTURED FILES: With the use of -input_mode and -output_mode, it is possible to specify the processing of unstructured (stream) files with copy_file. Three possibilities involving unstructured files exist: stream to record, record to stream, and stream to stream.

Stream to record copying involves reading input lines and writing them as records. Record to stream copying involves reading records, appending a newline character and writing these characters to the output stream. Stream to stream copying involves simply reading characters from the input stream and writing them to the output stream.

In stream to record and record to stream copying, -nnl inhibits the copying or addition of newline characters.

In copies involving stream input, -from, -to, and -count specify positions in terms of lines unless -character has been specified, in which case positioning is in terms of characters.

NOTES ON KEYED FILES: The copy_file command can copy a keyed sequential file to produce an output file that has been restructured for maximum compactness as a keyed file or as though it were purely sequential. By default, the command copies only records and does not place keys in the output

copy_file (cpf)

file. To copy the keys, the -keyed control argument must be used. When -keyed is used, the input file must be a keyed sequential file. Whether keys are copied or not, control arguments can be used to delimit the range of records to be copied (i.e., -start, -stop, -from, -to, -count). Copying is always performed in key order.

NOTES: The input and output files can be any combination of structured or unstructured files. The input file can be copied either partially or in its entirety.

If either the input or output specification is an attach description, it is used to attach a uniquely named I/O switch to the file. The switch is opened, the copy performed, and then the switch is closed and detached. Alternately, the input or output file can be specified by an I/O switch name. Either the io_call command or iox_ subroutine can be used to attach the file prior to the invocation of the copy_file command. (See the description of the io_call command in this manual and the iox_ subroutine in the MPM Subroutines.)

If the input file is specified by an I/O switch name and the switch is not open, the copy_file command opens it for (keyed_)sequential_input or stream_input, performs the copy, and closes it. If the switch is already open when the copy_file command is invoked, the opening mode must be sequential_input, sequential_input_output, keyed_sequential_input, keyed_sequential_update, stream_input, or stream_input_output. The switch is not closed after the copy has been performed.

The "next record" or "next byte" must be defined if neither the -start nor -from control argument is used to specify an absolute starting position within the input file. If the I/O switch is opened by the copy_file command, the next record is the first record of the file; otherwise, the next record is that record at which the file is positioned when the copy_file command is invoked. If the -character control argument has been specified and -from, -to, or -count has been specified and the input file is a stream, positioning is performed in terms of characters rather than lines or records.

If the output file is specified by an I/O switch name and the switch is not open, the copy_file command opens it for (keyed_)sequential_output, performs the copy, and closes it.

copy_file (cpf)

If the switch is already open when the copy_file command is invoked, the opening mode must be sequential_output, sequential_input_output, keyed_sequential_output, keyed_sequential_update, direct_output, direct_update, stream_output, or stream_input_output. (In update mode, output file records with keys that duplicate input file records are rewritten.) The switch is not closed after the copy has been performed.

The -from and -start control arguments are mutually exclusive. The -to, -stop, -count, and -all control arguments are mutually exclusive. The -nnl control argument is mutually exclusive with stream to stream or record to record copying. The -start and -stop control arguments are mutually exclusive with stream input. The -brief and -long control arguments are mutually exclusive. The informative message, printed by default, appears as one of the following:

```
  345 records copied.
  345 records read; 4002 characters written.
 4002 characters read; 345 records written.
 4002 characters copied.
```

EXAMPLES: To copy an entire file from an already attached file to the segment in_copy, type:

```
! cpf -isw in -ods "vfile_ in_copy"
```

To print the first 13 records of a tape file, type:

```
! cpf -ct 13 -ids "tape_ansi_ 887677 -name TEST21 -ret all"
   -osw user_output -ct 13
```

To copy 13 records from an already attached file to another already attached file, starting with the 56th record of the input file, type:

```
! cpf -isw in -osw out -from 56 -ct 13
```

To copy records 43 through 78 from an already attached file to an already attached file, type:

```
! cpf -isw in -osw out -from 43 -to 78
```

copy_file (cpf)

To copy all but the first seven records from segment testdata.11 to an already attached file, type:

```
! cpf -ids "vfile_ testdata.11" -osw out -fm 8
```

To copy an entire keyed sequential file with keys, type:

```
! cpf -isw in -osw out -all -keyed
```

To copy 13 records of a keyed sequential file starting with the record whose key is ASD66 to a sequential output file, the following line is typed. (No keys are copied.)

```
! cpf -isw in -osw out -sr ASD66 -ct 13
```

To copy the records and keys from a keyed sequential file up to and including the record whose key is bb"bb, type:

```
! cpf -keyed -isw in -osw out -sp "bb""bb"
```

To copy a tape_mult_ tape to another tape, type:

```
! cpf -ids "tape_mult_ m2156" -ods "tape_mult_ m2752 -write"
   -imode si -omode so
```

To copy each line in the segment test to a record on an output tape, type:

```
! cpf -ids "vfile_ test" -ods "tape_ansi_ 792561 -write
    -format vb -bk 4000" -imode si
```

To copy 20 records of an already attached keyed sequential file, starting with the record whose key is ASD60 to a stream output file without appending newlines, type:

```
! cpf -isw in -osw out -sr ASD60 -ct 20 -omode so -nnl
```

To copy 27 lines without their newline characters from an already attached stream file to an already attached output file, type:

```
! cpf -isw in -osw out -imode si -nnl
```

SYNTAX AS A COMMAND:

copy_iacl_dir path1_1_ path2_1_ {... path1_n_ path2_n_}

FUNCTION: copies the initial access control list for directories (directory initial ACL) of one directory to another, replacing the current directory initial ACL if necessary.

ARGUMENTS:

path1_i_
is the pathname of a directory. The star convention is allowed.

path2_i_
is the pathname of the target directory. The equal convention is allowed.

NOTES: See the MPM Reference Guide for a description of initial ACL's.

copy_iacl_seg

SYNTAX AS A COMMAND:

copy_iacl_seg path1_1_ path2_1_ {... path1_n_ path2_n_}

FUNCTION: copies a segment initial access control list (initial ACL) from one directory to another, replacing the current initial ACL if necessary.

ARGUMENTS:

path1_i_
is the directory from which the initial ACL is to be copied. The star convention is allowed.

path2_i_
is the directory into which the initial ACL is to be copied. The equal convention is allowed.

create (cr)

SYNTAX AS A COMMAND:

cr paths

FUNCTION: causes a segment to be created in a specified directory, or in the working directory. That is, it creates a storage system entry for an empty segment.

ARGUMENTS:

paths
are pathnames of segments to be created.

ACCESS REQUIRED: The user must have append access to a directory in order to create a segment in that directory.

NOTES: If the creation of a new segment would introduce a duplication of names within the directory, and if the old segment has only one name, the user is interrogated whether to delete the segment bearing the old instance of the name. If the old segment has multiple names, the conflicting name is removed and a message to that effect is issued to the user. In either case, since the directory is being changed, the user must also have modify permission for the directory.

The user creating the new segment is given rw access to the segment created.

All directories specified in paths must already exist. That is, only a single level of the storage system hierarchy can be created with this command.

If any one of the paths is the name of an existing link, a segment is created in the place specified by that link. The user must have append access to the directory containing the link target in order to create this segment.

See the description of the create_dir and link commands for an explanation of the creation of directories and links, respectively.

create (cr)

EXAMPLES:

The command line:

```
! cr first_class_mail >udd>Demo>Jones>alpha>beta
```

creates the segment first_class_mail in the working directory and the segment beta in the directory >udd>Demo>Jones>alpha. As explained above, the directory alpha must already exist.

create_data_segment (cds)

SYNTAX AS A COMMAND:

cds path {-control_arg}

FUNCTION: translates a create_data_segment (CDS) source program into an object segment. A listing segment is optionally created. These results are placed in the user's working directory. This command cannot be called recursively.

ARGUMENTS:

path
is the pathname of a CDS segment. If path does not have a cds suffix, one is asssumed. However, the cds suffix must be the last component of the name of the source segment.

CONTROL ARGUMENTS:

-list, -ls
produces a source listing of the CDS program used to generate the data segment followed by object segment information (as printed by the print_link_info command described in the MPM Subsystem Writers' Guide) about the actual object segment created.

NOTES: The source for create_data_segment programs is standard PL/I with the restriction that the program include a call to the create_data_segment_ subroutine. The create_data_segment_ subroutine creates a standard object segment from PL/I data structures passed to it as parameters. These data structures can be initialized with arbitrarily complex PL/I statements in the CDS program. (See the MPM Subroutines for a description of the create_data_segment_ subroutine.)

Since the create_data_segment command invokes the PL/I compiler to first compile the CDS segment, any errors that the compiler finds are reported by its standard technique. If any errors with a severity greater than 2 occur, the CDS run is aborted and an object segment is not created.

create_dir (cd)

SYNTAX AS A COMMAND:

cd paths {-control_args}

FUNCTION: causes a specified directory branch to be created in a specified directory, or in the working directory. That is, it creates a storage system entry for an empty subdirectory. See the description of the create command for information on the creation of segments.

ARGUMENTS:

paths
are pathnames of directories to be created.

CONTROL ARGUEMENTS:

-access_class STR, -acc STR
applies to each pathi and causes each directory created to be upgraded to the specified access class. The access class can be specified with either long or short names.

-logical_volume VOL, -lv VOL
specifies that each directory created is to be a master directory whose segments are to reside on the logical volume named VOL.

-quota N
specifies the quota to be given to the directory when it is created. This argument must be specified if either the -access_class or -logical_volume control argument is specified. If omitted, the directory is given zero quota. The value of N must be a positive integer, and applies to each pathi.

ACCESS REQUIRED: The user must have append permission to a directory in order to create a subdirectory in that directory.

NOTES: If a quota is specified and the directory being created is not a master directory, the containing directory must have sufficient quota to move quota to the directory being created. (See the move_quota command for additional information.)

If the creation of a new subdirectory introduces a duplication of names within the directory, and if the old subdirectory has

create_dir (cd)

only one name, the operation is not performed. If the old subdirectory has multiple names, the conflicting name is removed and a message to that effect issued to the user.

The user is given sma access on the created subdirectory.

All superior directories specified in pathi must already exist. That is, only a single level of storage system directory hierarchy can be created in a single invocation of the create_dir command.

In order to create a master directory, the user must have a quota account on the logical volume with sufficient volume quota to create the directory. A master directory must always have a nonzero quota; therefore, the -quota control argument must always be given when creating a master directory. A master directory can be created even though the logical volume is not mounted.

Each upgraded directory must have a quota greater than zero and must have an access class that is greater than its containing directory. The specified access class must also be less than or equal to the maximum access authorization of the process.

When the -access_class control argument is specified, the command does not create a new directory through a link. Creating through links is allowed only when the access class of the containing directory is taken as the default.

EXAMPLES:

The command line:

```
! cd sub >my_dir>alpha>new
```

creates the directory sub immediately inferior to the current working directory and the directory new immediately inferior to the directory >my_dir>alpha. As noted above, the directories my_dir and alpha must already exist. Both directories are assigned the access class of their containing directory.

create_dir (cd)

The command line:

```
! cd subA -access_class a,c1,c2 -quota 5
```

creates the directory subA with an access class of a,c1,c2 and a quota of 5 pages. The directory subA is created immediately inferior to the working directory. (The access class names a, c1, and c2 used in the example represent possible names defined for the site. See the print_auth_names command for more details on access class names.)

The command line:

```
! cd subB -logical_volume volz -quota 100
```

creates a master directory subB immediately inferior to the working directory. Segments created in this new directory will reside on the logical volume named volz. The directory subB is given a quota of 100 records.

cumulative_page_trace (cpt)

SYNTAX AS A COMMAND:

cpt command_line {-control_args}

FUNCTION: accumulates page trace data so that the total set of pages used during the invocation of a command or subsystem can be determined. The command accumulates data from one invocation of itself to the next. Output from the command is in tabular format showing all pages that have been referenced by the user's process. A trace in the format of that produced by the page_trace command can also be obtained.

ARGUMENTS:

command_line
is a character string to be interpreted by the command processor as a command line. If this character string contains blanks, it must be surrounded by quotes. All procedures invoked as a result of processing this command line are metered by the cumulative_page_trace command.

CONTROL ARGUMENTS:

-count, -ct
prints the accumulated results, giving the number of each page and the number of faults for each page. This control argument cannot be used with -print or -total (see "Notes" below).

-flush
clears primary memory before each invocation of the command line and after each interrupt. This helps the user determine the number of page faults but increases the cost.

-interrupt N, -int N
interrupts execution every N virtual CPU milliseconds for page fault sampling. The default is 500 CPU milliseconds.

-long, -lg
produces output in long format, giving full pathnames.

-loop N
calls the command to be metered N times.

-print, -pr
prints the accumulated results, giving the number of each page referenced. This control argument cannot be used with -count or -total (see "Notes" below).

cumulative_page_trace (cpt)

-print_linkage_faults
prints all accumulated linkage faults and calls to the hcs_$make_ptr entry point.

-reset, -rs
resets the table of accumulated data. If the table is not reset, data from the current use of cumulative_page_trace is added to that obtained earlier in the process.

-short, -sh
formats output for a line length of 80.

-sleep N
waits for N seconds after each call to the command being metered.

-timers
includes all faults between signal and restart.

-total, -tt
prints the total number of page faults, the total number of segment faults, and the number of pages referenced for each segment. This control argument cannot be used with -count or -print (see "Notes" below).

-trace_linkage_faults
accumulates linkage faults information along with page and segment fault information.

-trace path
writes the trace on the segment named path using an I/O switch named cpt.out; cumulative_page_trace attaches and detaches this switch.

NOTES: The cumulative_page_trace command operates by sampling and reading the system trace array after invocation of a command and at repeated intervals. Control arguments are given to specify the detailed operation of the cumulative_page_trace command.

The command line used to invoke the cumulative_page_trace command includes the command or subsystem to be traced as well as optional control arguments.

At least one of three generic operations must be requested. They may all be combined and, if so, are performed in the following order: resetting the table of accumulated data,

cumulative_page_trace (cpt)

calling the command to be metered, applying the specified control arguments, and printing the results in the specified format.

The default mode of operation uses interrupts for page fault sampling with a default sampling time of 500 milliseconds. If this figure is too large, messages indicate that some page faults may have been missed; a smaller value can then be chosen. The cost of a smaller value is high and may cause additional side effects. If the command or subsystem to be metered includes the taking of CPUT interrupts, then the -timers control argument should be specified. This control argument causes some of the page faults of the metering mechanism to be included as well.

Only one of the control arguments -print, -count, or -total can be specified. Each of these control arguments produces printed output in a different format. If more than one format is desired, the command must be invoked once for each format.

EXAMPLES:

The command line:

```
! cpt "pl1 test" -interrupt 400 -trace trace_out
```

calls the pl1 command to compile the program named test, requesting an interrupt every 400 milliseconds to obtain page trace information. Trace information is placed in a segment named trace_out.

The command line:

```
! cpt "list -pn >udd>Multics" -loop 2 -sleep 10
```

calls the list command twice, and sleeps for 10 seconds between calls.

The command line:

```
! cpt -print
```

prints the accumulated results of previous metering.

damaged_sw_off (dsf)

SYNTAX AS A COMMAND:

dsf paths

FUNCTION: resets the damaged switch for segments. See "Notes" below, for an explanation of the damaged switch.

ARGUMENTS:

paths
are pathnames of segments. The star convention is allowed.

NOTES: If a device error or system crash destroys a page of a segment, the supervisor turns on the damaged switch associated with the segment. Two cases of damaged segments can occur. Sometimes, the only valid copy of a page of the segment is destroyed; if so, a page of zeros appears in the segment. In other rare cases, a modified page of a segment cannot be written out due to a paging device error; when this occurs, the original unmodified page may be supplied.

An attempt to reference the contents of a segment whose damaged switch is on causes an error with the message:

Entry has been damaged. Please type "help damaged_segments.gi"

When a damaged segment is detected, the owner of the segment should change the access of the segment so that no other user can reference it, and then reset the damaged switch with the damaged_sw_off command. The owner should then inspect the contents of the segment to determine whether the segment can be re-created or needs to be retrieved.

Users can explicitly turn on the damaged switch of a segment. This is generally done to test recovery procedures. See the description of the damaged_sw_on command.

damaged_sw_on (dsn)

SYNTAX AS A COMMAND:

dsn paths

FUNCTION: sets the damaged segment switch on for a segment. The damaged switch is off for newly created segments. The switch can be turned on by the system (see "Notes" in the damaged_sw_off command for an explanation) or by a user conducting special tests.

ARGUMENTS:

paths
are pathnames of segments. The star convention is allowed.

NOTES: The damaged_sw_on command is primarily useful for testing recovery procedures. It should not be used instead of normal access control procedures for casual denial of normal access.

SYNTAX AS A COMMAND:

date {dt}

SYNTAX AS AN ACTIVE FUNCTION:

[date {dt}]

FUNCTION: returns the date abbreviation for a specified date or the current date.

ARGUMENTS:

dt
is a date-time in a form acceptable to convert_date_to_binary_. If no argument is specified, the current date is returned.

EXAMPLES:

The command line:

! date May 5, 1980

prints:

05/05/80

The command line:

! date Monday

prints the next occurrence of a Monday.

date_compiled (dtc)

SYNTAX AS A COMMAND:

dtc path {-control_arg} {components}

SYNTAX AS AN ACTIVE FUNCTION:

[dtc path {-control_arg} {components}]

FUNCTION: prints the date and time compiled and the compiler identifier for an object segment or an archive of object segments. For a bound object segment, the command prints the date and time compiled for each component.

ARGUMENTS:

path
is the pathname of an object segment, bound object segment, or an archive of object segments.

components
are names of components in a bound object segment or archive of object segments. If component names are specified, information on only these components is listed.

CONTROL ARGUMENTS:

-brief, -bf
lists the date and time compiled only (see "Examples" below).

-long, -lg
lists the date and time compiled, the segment name, the User_id of the person who compiled the segment, and the long form of the compiler identifier (see "Examples" below).

NOTES: If an archive is listed, the bind file is ignored.

If neither control argument is specified, the command lists the date and time compiled, the segment name, the User_id, and the short compiler identifier (see "Examples" below).

Invoked as an active function, dtc returns the first line of output that would be printed if it were invoked as a command.

date_compiled (dtc)

EXAMPLES: To check the compilation date of a private version of the list command in the working directory, type:

```
! date_compiled list -bf
  04/11/77 0922.2
```

To check information on the latest compilation of a Multics system installed command (that is unbound), such as demo_command, type:

```
! date_compiled >system_library_standard>demo_command
  03/09/77 1615.2 demo_command Martinson.SysMaint.a PL/I
```

To get compilation information on an entire bound object segment that is part of the standard Multics system, type:

```
! date_compiled >sss>bound_binder_
  Bound  10/26/77  1337.4  bound_binder_ Martinson.SysMaint.a
binder
  07/26/76 1048.4 bind Martinson.SysMaint.a PL/I
  10/26/77 1328.2 bx_ Martinson.SysMaint.a cds
                    .
                    .
                    .
  12/27/76 1354.3 old_make_bindmap_ Martinson.SysMaint.a PL/I
```

To get detailed information on one component of a bound object segment (in this case, bind in bound_binder_), type:

```
! date_compiled >sss>bound_binder_ -lg bind
  07/26/76  1048.4 bind Martinson.SysMaint.a Multics PL/I
    Compiler, Release 20e, of May 22, 1976
```

SYNTAX AS A COMMAND:

date_time {dt}

SYNTAX AS AN ACTIVE FUNCTION:

[date_time {dt}]

FUNCTION: returns a date and time value for a specified date-time or the current date-time consisting of: a date, a time from 0000.0 to 2359.9, a time zone, and a day of the week. The date and time value is returned as a single, quoted string of the form "mm/dd/yy hhmm.m zzz www" (e.g., "08/17/76 0945.7 est Tue").

ARGUMENTS:

dt
is a date-time in a form acceptable to conver_date_to_binary_. If no argument is specified, the current date-time is returned.

SYNTAX AS A COMMAND:

day {dt}

SYNTAX AS AN ACTIVE FUNCTION:

[day {dt}]

FUNCTION: returns a one- or two-digit number of a day of the month, from 1 to 31.

ARGUMENTS:

dt
is a date-time in a form acceptable to convert_date_to_binary_. If no argument is specified, the current day of the month is returned.

EXAMPLES:

The command line:

! day Friday

prints the one- or two-digit number of the next occurrence of Friday.

SYNTAX AS A COMMAND:

day_name {dt}

SYNTAX AS AN ACTIVE FUNCTION:

[day_name {dt}]

FUNCTION: returns the full name of a day of the week for a specified date or the current date.

ARGUMENTS:

dt
is a date_time in a form acceptable to convert_date_to_binary_. If no argument is specified, the name of the current day is returned.

EXAMPLES:

The command line:

```
! day_name 10/19/79
```

prints:

```
Friday
```

debug (db)

SYNTAX AS A COMMAND:

debug

FUNCTION: is an interactive debugging aid to be used in the Multics environment. It allows the user to look at or modify data or code. The user can stop execution of a program and examine its state by inserting "breakpoints" in the program before and/or during execution. A concise syntax for user requests, coupled with a complete system of defaults for unspecified items, allows the user to make many inquiries with little effort. Symbolic references permit the user to retreat from the machine-oriented debugging techniques of conventional systems and to refer to variables of interest directly by name.

The debug command uses a segment in the home directory to keep track of information about breaks. This segment is named Person_id.breaks, where Person_id is the login name of the user. The break segment is created if not found. If the segment cannot be created, the break features of debug are disabled and unusable.

Users who do not need the sophisticated machine level debugging provided by this command should refer to the probe command in this manual.

With the debug command, the user can:

Look at data or code;

Modify data or code;

Set a break;

Perform (possibly nonlocal) transfers;

Call procedures;

Trace the stack being used;

Look at procedure arguments;

Control and coordinate breaks;

debug (db)

Continue execution after a break fault;

Change the stack reference frame;

Print machine registers; and

Execute commands.

These functions are provided by two types of debug requests: data requests and control requests. The first five functions above are performed by data requests; the others, by control requests. Multiple debug requests (either data or control) can be placed on a line separated by semicolons (;).

Number Representation Conventions

Debug uses both octal and decimal representation of numbers. In general, machine-dependent numbers such as pointers, offsets, and registers are assumed to be octal, while counting arguments (e.g., specifying a source line number, printing the first 20 lines) and variables referenced by name are assumed to be decimal.

A decimal default can be changed to octal by preceding the number with the escape sequence "&o". An octal default can be changed to decimal by preceding the number with "&d".

Example	Description
x = 8	assign the value 8 to the program variable x. Program variables referenced by name are assumed to be decimal; if octal representation is preferred, type: x = &o10
$q = 77	assign the value of 77 to the q-register. Register values are machine dependent and assumed to be octal; if decimal representation is preferred, type: $q = &d63
/test/&a19	print line 19 of the source segment for test.
&a19,s8	print 8 source lines, beginning at line 19.

debug (db)

Data Requests

Data requests consist of three fields and have the following format:

<generalized address> <operator> <operands>

The generalized address defines the actual data or code of interest. It is ultimately reduced to segment number and offset by debug before being used. The operator field indicates to debug which function to perform, e.g., print or modify the data referenced by the generalized address. The operands field may or may not be necessary, depending on the operator. When these fields are specified, they are separated by blanks or commas.

When debug decodes a data request, it parses the generalized address and generates a pointer to the data being referenced. This pointer, called the working pointer, is changed whenever the generalized address is changed. It points into either the working segment, its stack frame, or its linkage section. The actual segment depends on the most recent specification in a generalized address. The form for a generalized address is as follows:

/<segment name>/<offset><segment ID><relative offset>

where each of the four fields is optional. The segment name is either a pathname, a reference name, or a segment number, and defines what is called the working segment. The segment ID specifies which of the data bases associated with the working segment is to be used in setting the working pointer. The segment ID can be one of the following:

&s
refers to the stack frame if the working segment is a procedure segment with an active stack frame.

&l
refers to an active linkage section (i.e., one with an entry in the linkage offset table (LOT) for the user's ring).

&t
refers to the working segment itself.

&a
refers to the source program for the working segment.

&p
refers to the parameters of an active invocation of a procedure.

&i
refers to an active internal static section (i.e., one with an entry in the internal static offset table (ISOT) for the user's ring).

The offset field is used as an offset within the segment referenced by the working pointer. For the working segment, this offset is relative to the base of the segment. If the working pointer points into an active stack frame, the offset is relative to the base of that frame. If the working pointer points into an active linkage section, the offset is relative to the beginning of that linkage section.

The offset can be either a number or a symbolic name. If a symbolic name is specified, a symbol table must exist for the working segment. See the translator commands for descriptions of symbol table creation. If a symbolic name begins with a numeric character, the escape characters &n (for name) must precede the name, to avoid interpreting the name as a number. For example:

```
/test/&n10&t
```

can be used in a debug request to specify the location associated with FORTRAN line number (i.e., label) 10.

The relative offset field allows the user to relocate the working pointer by a constant value or register. For example, a user wishing to reference the fourth word after the stack variable i he could use:

```
/test/i+4
```

as the generalized address. The relative offset can also assume the value of a register. For example, if the a-register contains the value 4 at the time of a break, then:

debug (db)

```
/test/100&s$a
```

sets the working pointer to offset 104 from the base of the stack frame. It is important to note that a + sign is not present when a register is used. (See "Registers" below.)

The three most common values for the segment ID field are &t, &s, and &l. These designate that the working pointer is to refer to, respectively, the working segment itself, its active stack frame, or its active linkage section. In addition, two other possible values of segment ID allow alternate methods of referring to locations in either the working segment or its stack frame.

A segment ID of &a refers to the ASCII source program for the working segment. Associated with this segment ID is a decimal line number, which must immediately follow the &a. This line number is used to generate a working pointer to the first word of code compiled for that line. A relative offset can follow the line number. Note that the line-number/code-location association can only be determined if a symbol table exists for the working segment. This example:

```
/test_seg/&a219+36
```

generates a working pointer that points to the thirty-sixth (octal) word in the text after the first word of code generated for line 219 in the source for the segment test_seg. If an offset field is given before &a, the offset is ignored. The offset of the working pointer is generated solely from the line number and the relative offset.

A segment ID of &p refers to the parameters of an active invocation of a procedure. If the current defaults specify an active stack frame, a number following the &p specifies the parameter that is to be addressed. The offset field is ignored, but a relative offset can be specified. This example:

```
/test_seg/&s;&p4+36,a14
```

causes the stack frame for test_seg to be the working segment, and the first 14 characters of the data contained at a location 36 words after the beginning of the fourth parameter are printed in ASCII format.

It is not necessary to specify all four fields of a generalized address. In fact, every field is optional. If a field is not specified, a default value is assumed that is frequently the last value that the field had. For example:

```
/test_seg/line&s+3
```

followed by the generalized address:

```
+4
```

is acceptable. The latter request is equivalent to:

```
/test_seg/line&s+7
```

One time that the defaults assumed are not the values of the previous data request is when a symbolic variable name or label is specified that causes some field to change. If this is the case, debug might recognize that the segment ID, for example, of the previous data request is not valid and set it appropriately. For example:

```
/test_seg/760&s
```

followed by:

```
regp
```

would cause the defaults to be changed to:

```
/test_seg/140&l
```

if regp is found at a relative offset of 140 (octal) in the linkage section. Note that the segment ID is changed to &l where it remains until explicitly or implicitly changed again.

Defaults are also reset to values different from the previous values when the segment name field is specified in a generalized address. In this case, the following actions are taken:

1) If the segment name begins with &n, take the rest of the characters composing the segment name and go to step 3 below, treating the string as a name. This convention allows the use of debug on segments whose names are composed of numeric characters.

debug (db)

2) If the segment name is really a segment number, this number is used in a search of all active stack frames to see if one exists for this segment. The search is from the highest stack depth (deepest in recursion) to the base of the stack so that if an active stack frame is found, it is the one most recently used. If an active stack frame is found, the generalized address defaults are set as follows:

a. working segment	the one specified by the given segment number.
b. offset	zero.
c. segment ID	&s, i.e., the working pointer points into the latest stack frame for the working segment.
d. relative offset	zero.

If no active stack frame is found, the defaults are set as above except that the segment ID is &t instead of &s, i.e., the working pointer points into the working segment itself.

3) If the segment name is a reference name known in this ring, the segment number for the segment being referenced is found, and then the defaults are calculated as if this segment number were given directly.

4) If the segment name is a pathname, the specified segment is initiated (it can already have been known) and the returned segment number is used as above.

5) If the segment name is of the form segname$entname, the stack is searched from the highest active frame (as in step 2) for the most recent frame associated with the entry point entname in the segment segname. The working segment becomes segname, and the remaining defaults are set as described in step 2.

The entire set of defaults that apply to a debug data request can be determined at any time by issuing the .d control request to print defaults. For the format and use of this request, see the description under "Control Requests" below.

Operator Field of Data Requests

After decoding the generalized address and determining the working pointer, debug checks the operator. The following

debug (db)

five operators are recognized:

1. , (comma) print
2. = assign
3. < set a break
4. > alter program control (i.e., "go to")
5. := call a procedure

If a debug request is terminated before an operator is encountered either by a semicolon or a newline character, the default operator used is ",", i.e., print. The one exception is that a blank line is ignored. The first, second, and fifth operators above have operands.

Print Request

For the print request, there are three optional operands. They are a single character specifying the output mode desired; a number indicating how much output is being requested; and a number in parentheses indicating the size of the output. The size has two meanings that are dependent on the output mode being used.

1) If the mode is comp-8 or comp-5, the size is the number of digits plus the sign, if present.

2) If the mode is not comp-8 or comp-5, the size is the number of bits to use in printing one item.

The size specification is permitted for the following modes: o, h, d, e, f, p, comp-5, comp-8. It is ignored for the following modes: i, l, a, b, comp-6, comp-7. All of the arguments are optional and spaces can appear between arguments. For example:

```
142&s,o(18)12
```

requests that 12 (decimal) half words starting at 142 (octal) in the stack be printed in octal format.

The following output modes are available for print requests (see "Output Modes" below for a full description):

o	octal
h	half-carriage octal
d	decimal
a	ASCII
i	instruction
p	pointer
s	source statement
l	code for line number
n	no output (just change defaults)
e	floating point with exponent
el	long floating point with exponent
f	floating point
fl	long floating point
b	bit string
g	graphic
comp-5	COBOL
comp-6	COBOL
comp-7	COBOL
comp-8	COBOL

The request:

```
+36,a14
```

requests that 14 (decimal) characters starting at 36 (octal) words after the current working pointer be printed in ASCII format. The output might be:

```
1416   1416  ">user_dir_dir>"
```